1995-09

Creation of a computer generated semi-autonomous entity able to function in an amphibious environment

Sobey, Thomas Jay.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

CREATION OF A COMPUTER GENERATED SEMI-AUTONOMOUS ENTITY ABLE TO FUNCTION IN AN AMPHIBIOUS ENVIRONMENT

by

Thomas Jay Sobey

September 1995

Co-Thesis Advisors: David R. Pratt
John S. Falby

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1995	Master's Thesis

4. TITLE AND SUBTITLE
CREATION OF A COMPUTER GENERATED SEMI-AUTONOMOUS ENTITY ABLE TO FUNCTION IN AN AMPHIBIOUS ENVIRONMENT

5. FUNDING NUMBERS

6. AUTHOR(S)
Sobey, Thomas J.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The problem is that current programs used to generate Semi-Automated Forces (SAF) are unable to fully simulate amphibious military operations in littoral regions. SAF development has focused on three separate paradigms: ground, air, and sea entities. Each set of entities has very different physical behaviors. For actions in littoral regions to be completely simulated, an entity must be able to cross between the sea and ground paradigms. This type of entity is necessary to simulate an amphibious assault.

The approach taken was to extend the ModSAF (Modular Semi-Automated Forces) program to include a "low-resolution" model of the Assault Amphibious Vehicle (AAV). The behaviors necessary to simulate an amphibious assault were also added.

The result of this work was the creation of a computer generated semi-automated entity able to function in an amphibious environment. In the 2D display of ModSAF, the physically based behavior of the entity was indistinguishable from a pure ground entity or a pure sea entity. Through comparisons with water and ground entities, the vehicle was shown to behave like a water vehicle in water and then to transition to, and behave like, a ground vehicle on land.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Semi-Autonomous Forces, Amphibious, Assault Amphibious Vehicles, AAV, ModSAF, Amphibious Landing, Littoral, Modeling, Simulation	82
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

CREATION OF A COMPUTER GENERATED SEMI-AUTONOMOUS ENTITY ABLE TO FUNCTION IN AN AMPHIBIOUS ENVIRONMENT

Thomas Jay Sobey
Captain, United States Marine Corps
M.A., Webster University, 1992
B.B.A., Texas A&M University, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1995

Author: 

Thomas Jay Sobey

Approved By: 

David R. Pratt, Co-Thesis Advisor

John S. Falby, Co-Thesis Advisor

Ted Lewis, Chairman
Department of Computer Science

ABSTRACT

The problem is that current programs used to generate Semi-Automated Forces (SAF) are unable to fully simulate amphibious military operations in littoral regions. SAF development has focused on three separate paradigms: ground, air, and sea entities. Each set of entities has very different physical behaviors. For actions in littoral regions to be completely simulated, an entity must be able to cross between the sea and ground paradigms. This type of entity is necessary to simulate an amphibious assault.

The approach taken was to extend the ModSAF (Modular Semi-Automated Forces) program to include a "low-resolution" model of the Assault Amphibious Vehicle (AAV). The behaviors necessary to simulate an amphibious assault were also added.

The result of this work was the creation of a computer generated semi-automated entity able to function in an amphibious environment. In the 2D display of ModSAF, the physically based behavior of the entity was indistinguishable from a pure ground entity or a pure sea entity. Through comparisons with water and ground entities, the vehicle was shown to behave like a water vehicle in water and then to transition to, and behave like, a ground vehicle on land.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

There are many people to whom I owe a debt of gratitude for their help in the completion of this work. First, I wish to thank Professor David Pratt and Professor John Falby for guiding me into the study of semi-autonomous forces. Second, I would like to thank the Marine Corps for providing the incentive and having the need for an accurate Assault Amphibious Vehicle. Thirdly, I would like to thank the Thai Hut for providing proper nourishment throughout the entire process.

Finally, I would like to thank my lovely wife, Terri, for the unwavering support and encouragement that makes all things possible.

I. INTRODUCTION

"The time always comes in battle when the decisions of statesmen and of generals can no longer effect the issue and when it is not within the power of our national wealth to change the balance decisively. Victory is never achieved prior to that point; it can only be won after the battle has been delivered into the hands of men who move in imminent danger of death." -- S.L.A. Marshall[FMFM95]

A. BACKGROUND

The question for military forces during peacetime is how to better train for war. The usual answer is to practice as realistically as possible, as often as possible. The problem is that practice is expensive in terms of equipment wear, supplies, ammunition, and occasionally, human life. Yet, to effectively simulate the "imminent danger of death", one must come perilously close to making it reality. The Department of Defense (DOD) has recognized that computer based simulations have the potential of providing the advantages of realistic training without the dangers.

The United States Marine Corps (a firm believer in putting the enemy in "imminent danger of death") has established the Marine Corps Modeling and Simulation Management Office (MCMSMO) to "provide a dedicated, service-wide, cross-functional activity to integrate and promote simulation-supported technology throughout the Marine Corps." [MCMS94] The effective use of simulation is looked upon as a way to safely and inexpensively train major subordinate command staffs.

B. PROBLEM

To create realistic simulations, there must be a large number of participants. Otherwise the complexity of a wartime situation, with its innumerable factors to consider, is lost. It is not feasible, nor reasonable, to expect that every participant must be a human connected into the simulation. Therefore, there must be a way to create a large number of combatants who behave in a realistic manner. One piece of the puzzle can be provided by semi-

automated, computer generated forces which are "entities" that have physically based behaviors, are able to execute assigned missions, and can react to changing enemy or terrain situations. These Computer Generated Forces (CGF's) will provide the bulk of the combatants in any realistic distributed simulation. A single operator can create and control many CGF's. The "smarter" the CGF's are, (i.e. requiring less human intervention) the more of them that can be controlled by a single operator. Since the eventual goal is over 300,000 entities involved in a simulation, relatively intelligent CGF's will be needed.

The United States Army has been working heavily on distributed interactive simulations for many years. A very large portion of the vehicles that the Army uses have been modeled already [CERA93a]. Only recently has work begun to create entities for Marine Corps specific needs. One of the vehicles that had yet to be modeled was the Assault Amphibian Vehicle (AAV) used exclusively by the Marine Corps to conduct amphibious assaults. The AAV is unique in that it can operate in two distinct environments: land and sea. The goal of this work was to create an AAV entity in the ModSAF (Modular Semi-Automated Forces)[CERA93a] program that demonstrated the ability to cross between the two environments while maintaining the proper physically based behaviors. The successful creation of this vehicle has proven the feasibility of simulating amphibious assaults within a distributed interactive simulation environment.

C. SCOPE OF WORK

A very large amount of work remains to bring Marine Corps (and Navy) forces into the realm of realistic modeling and simulation. This study focuses on one critical piece, the Assault Amphibious Vehicle used exclusively by the United States Marine Corps to project force ashore in a very up-close and personal manner. The vehicle has been created in ModSAF using as much previous work as possible. On shore behaviors closely resemble the M2 Bradley (as current Marine Corps AAV tactical teachings dictate) which is already modeled in the system. The AAV behavior in ship-to-shore movements is simplified to: (1) embark onto an amphibious warfare ship, (2) move to an assault position, (3) disembark the

static ship, (4) move into an appropriate assault formation (on-line or "wave"), (5) assault the designated beach, (6) secure the landing area by hastily occupying defensive positions, and (7) await a new mission (or continue executing already assigned ones).

D. ORGANIZATION

One goal of this study is to enable its use as a guide to creating entities in ModSAF. With that in mind, the organization of Chapters IV and V concentrate on the details of how the entity and its behaviors were modeled. Chapter II describes other work in the area of computer generated forces to give the user an idea of where this system will fit in. Chapter III describes the main program used for this study, ModSAF, in greater detail. Chapter VI discusses the new AAV designs being created and how the tactical behavior of the models will have to change. Chapter VII summarizes the work done and points out areas which will require further work.

II. PREVIOUS WORK

The need for computer generated forces (CGF's) is being driven by the Department of Defense at this time. A modern battlefield potentially has over 300,000 participants. While it is theoretically possible to have that many participants in vehicle simulators or on simulation consoles, computer generated entities are the only realistic answer.

The field of computer generated forces encompasses several disciplines. Some degree of Artificial Intelligence (AI) is often needed for an entity to behave and react in a realistic manner. Physically based modelling takes more computing power but has inherent advantages that make the entities more believable. The goal is for humans immersed in a virtual world to be unable to distinguish between entities which are generated and which ones have direct human control. The number of human participants and the computing power required for a large simulation requires networked interactions. The goal (which has succeeded on a smaller scale of ~2000 entities) is to have forces around the world practice in a single virtual world [TAMB95]. Some of the forces would actually be operating "in the field", some would be immersed in detailed simulators, some would be at computer consoles, and the majority would be computer generated. The "state" of the world needs to be updated anytime there is a change at any location which requires fast, real time networks. As an adjunct to the network, efficient storage of the "state" of the world for fast traversal and updates requires the latest database techniques. In addition to databases, the field of visualization is heavily involved, especially when trying to immerse the human into the simulation.

It would be a massive programming project to be able to do all of these well. Some of these projects are attempting just that. Unfortunately, these projects must decide their approach years before an actual product is available. Many of the smaller projects take novel approaches to one or two aspects of the overall problem. As a result, the best solution

will be to use individual "best" solutions to a particular aspect that can also interact through a common medium. For clarification, this chapter will discuss terminology used, other current and historical CGF projects with some of their advantages/disadvantages, and some local thesis work.

A. TERMINOLOGY

The government and military penchant for acronyms has resulted in a plethora of terms which mean the same thing. CGF's (computer generated forces) have already been mentioned. Other terms which have been used interchangeably are SAFOR (semi-automated forces), IFOR (intelligent forces), and AFOR (automated forces) [BOOK93]. A semi-automated force has some human control. The advantage is that one human can control a large number of SAFORs such as several platoons of tanks or humans. The interaction required is usually minimal, such as giving them an simple task and they accomplish it without any help. An IFOR or AFOR requires no human control and is much harder to simulate. For the remainder of this document, the term CGF will be used generically for all the above terms.

Another term that needs explaining is DIS. DIS can stand for *Distributed Interactive Simulations* in general or as the network protocol for communicating between simulators [VANB93]. The term DIS-compliant will be used to characterize a system which uses the network protocol. DIS should not be confused with DSI which is a *Distributed Simulation Internet*. DSI is the wide-area network which connects simulators participating at different sites together in one simulation [POPE91].

B. HISTORICAL AND CURRENT COMPUTER FORCE GENERATORS

The idea for autonomous or semi-autonomous agents has its roots in robotics and has been around for at least as long as science fiction writers. The ability to create an autonomous intelligent agent would be valuable in many areas of industry and science. The idea of using semi-autonomous agents to populate a virtual battlefield can be traced to the SIMNET (Simulation Network) project in 1982 [CERA93a]. Almost all major projects to

create computer generated forces have their basis in the work done for SIMNET. The following non-comprehensive survey of work in the area of computer generated forces (CGF's) should provide the reader an idea of the direction of research and approaches being taken.

1. SIMNET

Any discussion of computer generated forces must begin with SIMNET. The development of a large-scale network of interactive simulators to train U.S. Army ground and air vehicle crews was started in 1982 by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army. The resulting system proved the feasibility and applicability of using a Distributed Interactive Simulation for training, research, and development [VRAB92]. The need for CGF's comes from the cost factors that limit the simulation to about 1,000 human beings [TAMB95]. Since a realistic exercise that can train the upper echelons of command would require many more participants, the remaining entities must be CGF's. Not only must the CGF's be controllable by a few humans, they must also behave in such a manner that the other participants cannot distinguish between the humans and the CGF's.

SIMNET was intended to train individuals as part of M1 tank or M2 Bradley crews operating in simulators. Other participants could be at terminals or actually in real vehicles in the field. The system was also designed to be easily extensible for other simulators and other CGF's. The software that SIMNET used for its CGF's is described next.

The advantages of SIMNET are its interaction with networked simulators and the developmental work in visual simulation. The main disadvantages are the cost of the simulators and the development software. Many components of the system are special purpose devices that are difficult to change. Additionally, the need for completely automated forces with efficient AI ability has not been addressed. The entities modeled are relatively limited, as are the command/control communication structures. The project was

limited to providing lower level training vice higher command training and it has accomplished that purpose.

2. BBN SAF 4.3.3

Bolt, Beranek, and Newman Inc. (BBN) SAF 4.3.3 is the latest release of the original SIMNET-compatible software that generates forces for the SIMNET distributed interactive simulations. The majority of the software and development team are now part of Loral Advanced Distributed Simulation (LADS), Inc.

The software runs on Silicon Graphics and MIPS machines. It is written in Kernigan and Ritchie (K&R) C (a non-object-oriented language) using an object-oriented design. Like all SIMNET-compatible software, it runs over Ethernet using a DIS "like" protocol. Much of the DIS protocol was based on the work done for SIMNET. The main emphasis is close combat and it has entities such as ground maneuver units (armored, mechanized), dismounted infantry, fixed wing aircraft, and rotary wing aircraft. Units can be simulated and controlled from the individual level to the battalion level. A single work station can control up to 50 entities.

BBN SAF uses the same terrain database representation as ModSAF, which will be described in detail in Chapter III. The interface is an X-based two-dimensional display with a mouse/keyboard interface. An operator controls entities through the use of Combat Instruction Sets (CIS's) and Tactical Emergency (TAC/E) instruction. The operator defines CIS's to control each entity's reaction to battlefield events. The operator does not trigger the reactions, only battlefield situations trigger the reactions. The operator uses the TAC/E to temporarily override the CIS parameters. [BOOK93]

The main advantage of this system is that it builds upon the successes of SIMNET while using a more general software solution. This portion of the software only generates a 2D display. In SIMNET, another piece of software provides the 3D visualization. The interface is rather cumbersome, with many menus needed to create/control entities.

3. ODIN SAF

Odin SAF, named for the Norse god of war and wisdom, was sponsored by DARPA and developed by BBN in close cooperation with the Army's Topographic Engineering Center (TEC). It leveraged on the SAF technology developed for SIMNET. It can be considered the second generation of BBN SAF. The Battle of 73 Easting was recreated using the improved capabilities of Odin SAF. The advantages and disadvantages of this system are very similar to BBN SAF due to it being an evolution of that software. [BOOK93]

4. ModSAF

ModSAF (Modular Semi-Automated Forces) is currently up to version 1.5.1 and subsumes all the functions of BBN SAF and Odin SAF. It, like BBN SAF and Odin SAF, has been created by the development team at LADS (formerly BBN). It was developed for the IFOR/WISSARD (Intelligent Forces / What If Simulation System for Advanced Research and Development) program with additional support from the United States Army STRICOM's (Simulation Training and Instrumentation Command) ADST program (Advanced Distributed Simulation Technology). The basic idea and architectural construct is the same as BBN SAF and Odin SAF, however the programming construct is much more modular and therefore much easier to extend. The program is discussed in detail in Chapter III.[BOOK93]

5. BDS-D CGF

BDS-D stands for Battlefield Distributed Simulation - Developmental. This program is also being developed by Loral under the Army STRICOM ADST program. It is based upon the Simulated Warfare Environment Generator (SWEG), which is discussed below. The software for modeling entity behavior is being developed by BDM International, Inc. The long term goal for this software is to expand the functionality of current SAF's to include combat support elements and combat service support elements, to better represent

dynamic environmental effects on entities behaviors, and to better represent command and control across multiple echelons.

Entity behavior is modeled using the Action Cognitive Behavior Model (ACBM). The behavior is controlled by external data files which allows great flexibility during a model run. Network communications are not DIS-compliant and are not currently planned to be. The implementation of ACBM code was translated into C from the SWEG Fortran software. It was scheduled to be translated into C++ at some point. The system runs on a Sun or Silicon Graphics Workstation. The user interface is standard X-Window Motif and uses the paradigm of the Five Paragraph Order (OPORD) to control entities. The human operator gives and receives OPORDs and can issue and receive reports. [BOOK93]

The main advantage of this system is the environmental effects and the ability to add models and behaviors using external data files. The big disadvantages are its non-DIS-compliance and the original Fortran architecture. Even though it is being translated into C++, the design cannot take full advantage of new programming paradigms.

6. IST SAF

The Institute for Simulation and Training (IST) has been tasked by STRICOM to build a low cost computer force generator. They designed a system that can run on an Intel 386 machine under MS-DOS. The development language is ANSI C with a C++ compiler for strong type checking. Behaviors are encoded as states within a slightly modified Finite State Machine (FSM) that runs under an "executive" due to the non-preemptive nature of MS-DOS. To construct behaviors, a C programmer must work with a Subject Matter Expert (SME) and carefully design the state to not monopolize the system for too long. [BOOK93]

Communication across the simulation network is run on Ethernet but is not DIS-compliant. Entities are semi-dynamically load balanced across available simulation engines and tests have shown the system to degrade gracefully under load. The database format used is the standard SIMNET polygonal terrain database. All the terrain reasoning, Line of Sight, and route planning calculations are done based on polygons. This should

make an easy transition to dynamic terrain. The user interface is a 2-D plan view display using a mouse. [BOOK93]

The main advantages of this system are the low-cost hardware and the system architecture. The disadvantages are the non-DIS-compliance, the non-preemptive operating system, and the relatively low CPU power. This approach deserves special attention due to the advances in PC industry which will solve many of these problems while maintaining the low-cost advantage.

7. IFOR/WISSARD

Intelligent Forces / What If Simulation System for Advanced Research and Development (IFOR/WISSARD) is an ARPA program to improve autonomous forces for the Navy Tactical Air domain. It is funding three development efforts: (1) ModSAF (as discussed in Chapter III), (2) The Concurrent Control (CoCo) SAF by Hughes Research Lab (HRL), (3) and software based on SOAR (the name is taken from the software's architecture cycle of taking a State, applying an Operator, And generating a Result) by a consortium of universities working together (University of Michigan at Ann Arbor, Carnegie Mellon University, and ISI). [BOOK93]

ModSAF is used to control and visualize the entities. Both CoCo and SOAR provide the higher level controls. CoCo uses an arbitration scheme to resolve conflicts between multiple orders and has, so far, resulted in improved route planning capabilities. It can also be used together with case-based tools for rapid acquisition of tactics. SOAR originated in 1982 as an AI system and is concurrently used to model human behavior. It "is an attempt to create a general, uniform architecture to support all the capabilities necessary for general intelligent behavior, such as knowledge representation, problem solving, planning, learning, natural language understanding, and interaction with dynamic environments." [BOOK93] With ModSAF as the interface and low-level entity controller, CoCo and SOAR should provide very realistic high-level behaviors. [BOOK93]

This is not one specific software solution, but a combination of several solutions. The advantages of this system are the capabilities of ModSAF with the AI abilities of CoCo and

SOAR. Each of the systems uses DIS protocol to communicate and control. The next step would be to combine these systems with a networked 3D visualization system, such as NPSNET (Naval Postgraduate School Networked Vehicle Simulator), and some type of command/control simulator. The disadvantage of this solution is the need to coordinate capabilities across software packages.

8. CCTT SAF

The Close Combat Tactical Trainer (CCTT) program to develop the next generation of Army trainers is led by a team from Loral Federal Systems. The CGF developer is Science Applications International Corporation (SAIC). The emphasis is on representing the platforms that form an Army battalion to include; logistics and engineering elements, fire support of RAG/DAG's, rotary-wing aircraft, fixed-wing aircraft, and air defense platforms. [BOOK93]

9. SWEG/SUPPRESSOR

The Simulated Warfare Environment Generator (SWEG) was derived from the SUPPRESSOR system, originally developed for the Air Force by the Calspan Corporation from 1978 to 1981. It is a mission level, discrete event simulation system useful for evaluating electronic combat systems, weapon systems, or tactics in many-on-many scenarios. SWEG is an incremental modification of SUPPRESSOR version 5.1 to allow it to interoperate in real-time external hardware and software components. The modification was done by BDM International Inc., the same company developing BDS-D CGF described above. [BOOK93]

The software is written in Fortran, however, no assumptions about the entities or the simulation are made at all. Everything depends on the input data to include: the scenario, the platforms and systems, tactics, terrain, rules of engagement, command and control, and the number of sides in the conflict. This completely data driven scenario makes the system extremely flexible and makes it more of a programming environment rather than a combat modeler.

The terrain is derived from DMA (Defense Mapping Agency) terrain files and converted into small binary files for generation of a continuous surface defined by a variable-resolution triangular lattice. Other databases are used to contain information on each player (TDB - Type Data Base), and the scenario (SDB - Scenario Data Base). The user interface is batch processing with the results given as statistical models. SWEG is capable of displaying a graphical view of the scenario as it executes.

The advantages of this system are the data-driven flexibility of each scenario and the ability to use standard terrain data. The disadvantages are that it was designed for gathering statistical results vice visualization and that it was written in Fortran with a software architecture from 1979.

10. Janus

Janus was originally built by Lawrence Livermore National Labs to be an entity-level simulator. It has been transitioned to TRAC-White Sands for continued development. The individual entities (weapon systems, vehicles and soldiers) interact in the combat environment with values derived from real-world tests. The software is written in Fortran with some C graphics code. It is not completely DIS-compliant but the Naval Postgraduate School and Rand Corporation have done a proof-of-concept project called JLink (Janus Linked to DIS) to show that it can be done. The system has been ported to Sun and HP Workstations and the Army is looking to rewrite it in an Open Systems environment. [BOOK93]

The terrain data is provided by the Defense Mapping Agency and stored in two files: one for digitized terrain information, the other for contour lines, grid squares, vegetation, roads, built-up areas, and rivers. The user controls units through preset plans or directly during a scenario. The interface is a 2-D display using a mouse.

The advantages of this system are physically-based simulation of entities and the terrain reading ability. The disadvantages are the old Fortran architecture and the non-DIS compliance.

C. LOCAL THESIS WORK

1. Autonomous Agents

The Computer Science Department at the Naval Postgraduate School developed and implemented a three-dimensional simulation system -- NPSNET -- using the Distributed Interactive Simulation (DIS) protocol. Soon after its development, research work began on how to incorporate autonomous agents into this three-dimensional world. Two local theses, "NPSNET: Physically Based, Autonomous, Naval Surface Agents" written by LT John Hearne [HEAR93], and "Tactical Decision Making in Intelligent Agents: Developing Autonomous Forces in NPSNET" by CPT Michael Culpepper [CULP92], provided groundwork in the area of using an external planning agent to provide realistic behaviors to autonomous agents in NPSNET.

Both Hearne and Culpepper used an expert system tool -- CLIPS -- to develop expert system shells to replicate behaviors of computer agents in a three dimensional world. LT Hearne added intelligent, autonomous naval surface ships that incorporated the complexities of actual ship turning and propulsion dynamics. CPT Culpepper added autonomous armor units that planned target selection, subordinate missions, and the cooperative efforts of platoon sized elements. Both developments provided autonomous agents that react to a changing environment by using expert system rule sets. [HEAR92] [CULP92]

2. ModSAF Extensions

When LADS released ModSAF 1.0 in December 1993, two Naval Postgraduate School masters thesis students started working on extending its capabilities. Major Gary M. McAndrews, USA, addressed the problem of adding company-level missions to increase the number of forces a single operator can control. ModSAF 1.0 only had the capability to control platoons or individuals. The result of his work was the proof-of-concept company-level mission "Occupy an Assembly Area" that can be assigned to a company [MCAN94]. Another student, Major Howard L. Mohn, USA, added the ability for users to utilize a

standard Five Paragraph Order (OPORD) to implement mission planning and task assignment. Through the graphical user interface, a user can input an OPORD and have the system generate the ModSAF phases [MOHN94].

D. SUMMARY

The best solution will probably come from an approach similar to the IFOR/WISSARD program. Their results pick the "best of breed" for AI, physically based simulation and visualization, and command/control functions. The solution will have to use the DIS protocol for interfacing and controlling entities. The DIS protocol is the current and future of networking virtual worlds. It has some problems, but the standard is evolving and is accepted by almost all large systems in development. There is no one solution for creating CGFs. The size of the problem will require the combination of many disciplines and the constant evolution of software as the computer industry continues its exponential growth in capabilities.

III. ModSAF DESCRIPTION

A. DESCRIPTION

An excellent overview of the ModSAF system can be found in the papers by Calder, Smith, Courtemanche, Mar and Ceranowicz, *ModSAF Behavior Simulation and Control* [CALD93], Andrew Z. Ceranowicz, *ModSAF and Command and Control* [CERA93b], and Gary M. McAndrews, *Autonomous Agent Interactions in a Real-Time Simulation System* [MCAN94]. A basic overview of the ModSAF system and some of the particular ModSAF terms are defined in this chapter to aid in understanding the development of the Assault Amphibious Vehicle (AAV) entity.

1. Introduction

The Modular Semi-Automated Forces (ModSAF) system is a Distributed Interactive Simulation (DIS) system that portrays Computer Generated Forces (CGF) with realistic individual and unit behaviors. The system was originally sponsored by the Advanced Research Projects Agency (ARPA) WISSARD (What If Simulation System for Advanced Research and Development) project. ModSAF has become the standardized simulation platform for continued research in the use of Computer Generated Forces by the U.S. Army Simulation, Training, and Instrumentation Command's (STRICOM) Advanced Distributed Simulation Technology (ADST) program. ModSAF is a modular implementation of previous work conducted in SIMNET under DARPA and ODIN [CALD93].

2. Overview

ModSAF is object oriented standard Kernigan & Ritchie C code. The first release, 1.0 in December 1993, consisted of over 150 library modules and 215,000 lines of code. ModSAF 1.4 was released in January 1995 and is over one-half million lines of code [COUR95]. The current version is 1.5.1. It runs on SGI, Sun, MIPS, and IBM RISC 6000 Hardware systems [ROBA94b].

The architecture for the system includes three components: The ModSAF Command Workstation (SAFstation), the ModSAF Simulator (SAFsim), and the ModSAF Logger. The SAFstation provides the graphical user interface to the operator. The operator can create and place units, assign missions, and observe the execution of the units from this station. The operator is given a two dimensional view of the terrain database on which the exercise is being simulated. The SAFsim simulates the vehicles and units created by the operator. It provides realistic behaviors to the entities. The Logger is a recorder that records the states of the Persistent Object Database and the DIS database, and can replay exercises. [CERA93b]

3. Communications

The Distributed Interactive System concept is defined in Proposed IEEE Standard Draft, "Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications". "DIS is a time and space coherent synthetic representation of world environments designed for linking the interactive, free play activities of people in operational exercises."[STAN93] In a very general sense, computers can share information about the world (or terrain) and the positions and activities of the entities in the world, with each computer portraying the world and all its entities simultaneously. The DIS Protocol is a standardized format to package the necessary information about the entities being displayed in a DIS environment. ModSAF communicates with other computers in the simulation by sending and receiving DIS Protocol Data Units (PDUs).

Whereas the DIS protocol provides a way to share the physical state of the world between computers, ModSAF has its own Persistent Object (PO) Database to retain information about the entities it is simulating. The PO Database keeps track of information about a unit including the unit's current mission and status, the unit's organization, and the individual vehicle information for each vehicle in the unit. The ModSAF computers share command and control and system information via the Persistent Object (PO) Protocol. [CERA93b]

ModSAF maintains two databases (see Figure 1): the DIS Database, and the PO Database. The DIS Database is a conceptual database, used to share information between ModSAF and external DIS simulation systems. The PO Database stores internal information about the world and its entities.

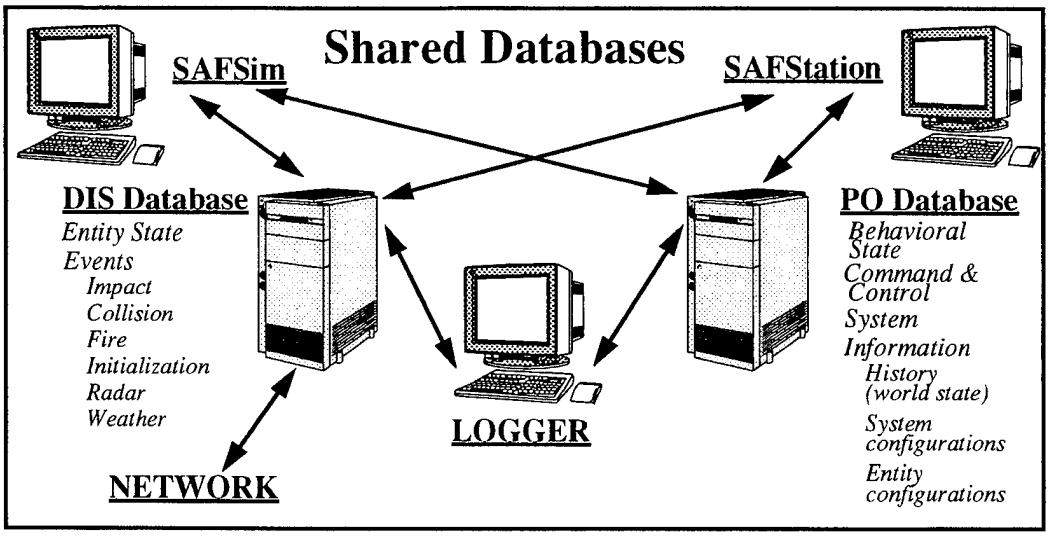

Figure 1: From [ROBA94a] Shared Databases

B. ModSAF SOFTWARE ARCHITECTURE

Since ModSAF was written in K & R C code, it is not object-oriented in the purest sense of the definition. The ModSAF system replicates the behavior of classes and methods offered in C++ by utilizing layering and object-based programming techniques.

> "Layering is a design methodology in which software modules are grouped into layers, and software in one layer is restricted to use only functions and services available in lower layers.... Object-based programming techniques are used to cleanly separate the subsystems or modules into classes of objects. Each object class is defined by a data structure and a family of functions which operate on that data structure." [CALD93]

The category of software modules that the company assembly mission utilizes falls into the set of simulation modules. This set of modules provides the behaviors for the ModSAF entities. Figure 2 lists some of the several different ModSAF library modules. The first set of modules are behavioral modules. These library modules provide the

behaviors for a vehicle or the components of a vehicle. The turret, hull, and guns of a tank are each controlled by an individual behavior module. Libraries prefixed with "libv" are vehicle simulation modules. These modules perform task actions for individual vehicles. Libraries prefixed with "libu" are unit simulation modules. These modules provide the task

Component Dynamics	libgenturret libmissile libturrets	libguns libmlauncher	libhulls libradar	libifdam libtracked
Unit Level Control	libuactcontact libubingofuel libuenemy libumount libutargeter	libuassault libucap libuflwrte libuoccpos libutraveling	libuataint libucommit libuflyrte libuoverwatchmove libuebkpckup	libuatgrndtrgt libudsmnt libuhalt libupoccpos
Vehicle Control	libvassess libvcollide libvflygrndavoid libvmount libvspotter libvterrain	libvataint libvecmat libview libvmove libvtab libvembark	libvatgrndtrgt libvenemy libvisual libvorbit libvtakeoff	libvcap libvflwrte libvland libvsearch libvtargeter

Figure 2: ModSAF Library Modules

actions for entire units. This is accomplished by the layering and object-based programming techniques of the ModSAF system.

C. ModSAF COMMAND AND CONTROL

The architecture selected to replicate command and control of the computer generated forces provides capabilities, identified as command and control goals, to the developer [CALD93]. These goals include:

- The capability to create complex missions including preplanned contingency operations,
- The altering of a mission after assignment,
- An operator's ability to override the simulation at any time for any unit,
- A defined architecture for unit and individual behaviors,
- A graphical user interface with available missions,
- A defined structure to explain unit and individual behaviors to the user.

These capabilities allow the user to have finely tuned control of a unit or to have minimal supervision of a unit during its mission execution.

1. ModSAF Objects and Entities

An object in ModSAF is stored in the PO Database. Objects may include graphical control measures for a particular unit, individual vehicles, or entire units. Objects which are simulated by the SAFsim are termed ModSAF entities.

> "When a SAFsim simulates a unit, the SAFsim not only creates the SAF entities (such as a plane) in a unit but also builds a structure corresponding to the unit hierarchy. The user can then issue commands to the top-level units or drop down the chain of command to give orders to subordinate units or vehicles. The SAFsim interprets these orders and then generates the appropriate unit and vehicle behavior and tactics without further action from the user. However, the user can override or interrupt any automated behavior." [MODS94]

The behaviors of units and vehicles is controlled by "tasks" and "task frames". "A task is a behavior performed by a ModSAF entity or unit on the battlefield.... Task frames group a collection of related tasks that run at the same time." [MODS94] A more detailed description of tasks and task frames is given in the following sections.

2. Tasks

> "The foundation of the ModSAF command and control architecture is the concept of a task. Most tasks are behaviors performed by units or individuals on the battlefield, and are used by ModSAF to model the information processing done by its simulated entities.... There are five types of tasks which are implemented in the ModSAF system: unit tasks, individual vehicle tasks, reactive tasks, enabling tasks, and arbitration tasks." [CALD93]

An individual vehicle task controls the lowest-level actuators of a vehicle. These actuators control the simulated capabilities of checking intervisibility, target detection, target identification, target selection, fire planning, collision avoidance, and detection [MODS94]. The vehicle tasks take inputs from its sensors and other actuators and produce commands for the physical actuators [CERA93b].

A unit task encapsulates the behaviors of a unit and its individual vehicles for a particular task. For example, in the ModSAF unit task UTravel (a unit travelling task), the

task issues individual vehicle tasks to each member of the platoon, and monitors the collective status of the unit's movement. Changes to the current situation or parameter changes by the operator are handled by the unit task, which may introduce new vehicle tasks, or terminate individual vehicle tasks that have ended. The concept of issuing lower-level tasks from the unit level task models the military command and control structure [CERA93b]. For example, a platoon leader receives an order from his company commander to move to a location. The platoon leader develops a plan that will collectively get his unit to that location in a given formation. He then issues the orders to the individual vehicles to accomplish this mission. ModSAF replicates this command architecture through the use of unit level tasks.

A reactive task is triggered in response to a change in the environment. Similar to a platoon drill, the reactive task is a pre-defined set of reactions that a unit or vehicle will implement in response to a specific environmental change. For example, when assigning a unit a move task, the operator can set specific parameters of how to react to an enemy force. The reactive tasks for "Actions on Contact" can be set by the operator as parametric inputs. The operator can select the enemy vehicle thresholds and the resulting action by the unit.

Suppose the operator decides that if the unit comes under fire by less than three vehicles it should assault them, but if there are three vehicles or more it should occupy a defensive position. Also, if the unit is not being fired upon, and there are more than three enemy vehicles, conduct a contact drill. If there are less than three enemy vehicles do nothing. These decisions can be input into the parametric entry graphical user interface. The results of the operators inputs are shown in Table 1. The result is the appearance of decision making at the lowest level.

Enemy Vehicles \ Situation	Under Fire	Not Under Fire
Less Than Three Vehicles	Assault	No Action
Three or More Vehicles	Occupy Position	Contact Drill

Table 1: Action on Contact

Enabling tasks link the execution of the task frames. They are defined within the mission. Example enabling tasks include continue (continue with the next task frame when the current one ends), on order (after the previous task frame ends, halt execution until the operator says to continue), control measure (when a unit hits a graphical control measure change task frames), or at a certain time execute a task frame. Enabling tasks give the unit alternative actions to take in response to events the mission developer has foreseen when developing the mission [CALD93].

3. Task Frames

Related tasks which run concurrently to accomplish an action are termed "task frames". A task frame represents a phase of a mission and are defined with associated parameters. Some of these parameters may be adjusted by the operator to modify the behavior during the frame [CERA93b] [CALD93]. For example, a platoon conducting a move operation is operating within a task frame. The task frame includes the movement task and the reactive tasks for "Actions on Contact". The reactive parameters may be set by the operator when assigning the move task frame. As the move task is operating, the platoon may encounter an enemy unit. The move task will be pushed on a stack while the reactionary, contact drill, assault, or occupy position task executes. The operator can stop, change, or override the reaction of the unit during the execution of the move task frame. "Task frames are typically composed of move, shoot, and react tasks." [MODS94]

4. Missions

A sequence of task frames collectively form a mission. Before the next frame begins, the previous frame must have ended. An example mission which includes several separate tasks would be a platoon assigned to move along a route, attack an objective, and then move to another location and occupy a position. The attack on the objective will not begin until the movement along the route has ended. [CALD93]

ModSAF provides platoon level missions for ground vehicle platoons and dismounted infantry platoons. Some platoon missions may be assigned to a single vehicle. These

missions include: "Move", "Follow a Vehicle", "Occupy Position", and "Assault". One mission written solely for a ground vehicle platoon is "Bounding Overwatch." In a Bounding Overwatch mission, the platoon is split into two sections. One section moves and the other section stops to cover its movement. Assigning this mission to a single vehicle would not be in context with the mission parameters. The dismounted platoon missions include "Mount" and "Dismount." A platoon must be a dismounted infantry platoon to perform these missions.

5. Task Manager

The collection of tasks and task frames which combine to form missions are controlled by the ModSAF Task Manager. The task manager maintains information about prerequisite tasks and follow-on tasks for every mission. For example, a helicopter is given a move mission. A prerequisite task would be to "take-off" and a follow-on task may be to "land". The task manager maintains these task dependencies and develops a task execution list that considers the required before and after tasks along with the specified task. It then executes the tasks in the task execution list order. The task manager also handles the task frame management for the unit and vehicle tasks. [CALD93]

6. Task Arbitration

Often, more than one task is operating at the same time. These tasks may offer different commands to the vehicle. In the previous example of a platoon performing a movement that encounters an enemy force, the move task is setting the move path for each individual vehicle. Suddenly, a reactive "Action Drill" is initiated in response to making contact with the enemy. The "Action Drill" task will also give movement paths to each of the individual vehicles. The vehicles are given two possibly unique paths to follow, and some method of deconfliction is needed. It is the responsibility of the Task Arbitrator to decide which movement path to utilize for each vehicle. The Task Arbitrator takes all recommendations for the control of a particular actuator (movement, fire control, sensors) and then decides, based on a priority scheme, which task will control that actuator [CALD93].

D. FINITE STATE MACHINES

ModSAF's unit tasks, vehicle tasks, and some behavioral tasks are implemented using an "Augmented Asynchronous Finite State Machine" format. The tasks are developed by separating them into states of a finite state machine. Once the finite state machine (FSM) is coded, a "finite state machine to C code" conversion utility is called to convert the FSM to standard K & R C code. The format is asynchronous in that units may generate outputs in response to a particular event or group of events, and is augmented in that not only does it keep track of the state of a unit/entity, but it also maintains additional information (in the PO database) of other private variables besides just the state variables. [ROBA94a]

E. INTERACTION WITH ModSAF

The ModSAF system can be modified and extended by others to support multiple behavioral representations and multiple levels of command and control. There are several different ways to interact with ModSAF. A developer can change existing software modules or add new software modules, replace entire subsystems, or write separate programs that communicate with ModSAF through the Persistent Object Database [CALD93]. The AAV was created by adding it to the physical database, defining its weapons, ammunition, amphibious ability, and new behaviors (such as ship-to-shore movement).

F. SUMMARY

The ModSAF system provides low-level realistic behavior for CGF's across a DIS system. The modular software architecture of the system is designed to be extensible and flexible. The GUI interface allows one operator to have high-level control over many entities. The shared database paradigm will allow the scalability required for large-scale battlefield simulations. These capabilities are why ModSAF is the standard simulation platform for continued research in the use of CGFs by STRICOM's ADST program.

IV. ASSAULT AMPHIBIOUS VEHICLE

A. OVERVIEW

The simulation entity able to cross paradigms, from land to water (and back), is as unique in the simulation arena as it is in the real world. The only vehicle capable of being launched from a ship into the water, assault a beach, and continue on to conduct operations in the desert is the Assault Amphibious Vehicle (AAV) as shown in Figure 3. At the time of this study, no ModSAF vehicles are able to behave in this manner.

The modeling of operations in littoral regions, a specialty of the United States Marine Corps, requires that this unique vehicle be accurately simulated. The construct of ModSAF allows vehicles to be added fairly easily. It takes a physical description of each vehicle and creates the 2D model which behaves within the specified parameters. The end result is an AAV that is the proper size with respect to all other entities, has the correct armament, can climb a 60 degree obstacle, can move at 65 kmh (kilometers per hour) on land and 12 kmh in water, will slow down in rough terrain, and will only drive until its fuel runs out.

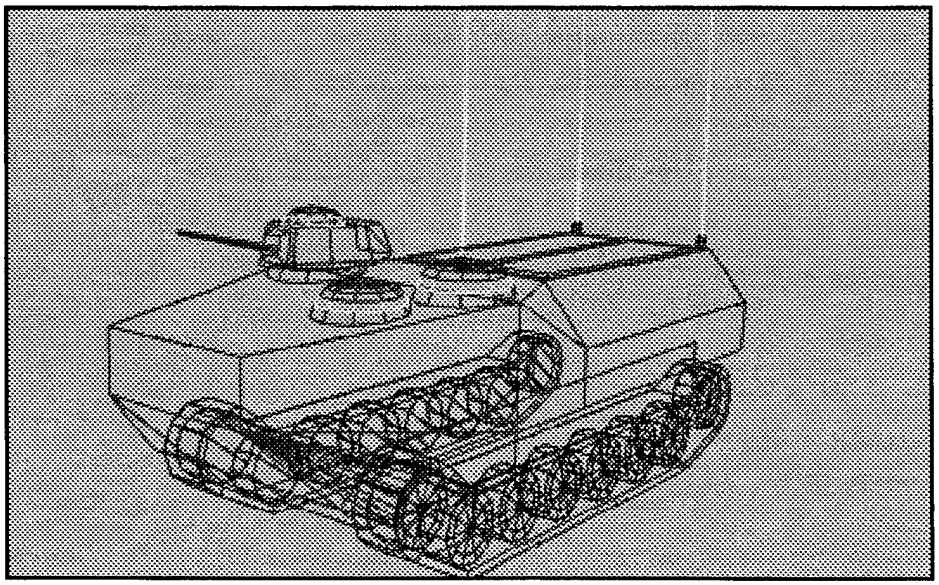

Figure 3: AAVP7A1

The steps necessary to create the AAV in ModSAF are to be discussed later in the chapter. An amphibious assault ship was also added to allow the demonstration of a limited amphibious landing. Its limitations and minimal abilities will also be discussed later.

B. BRIEF HISTORY

The idea of attacking your enemy from the sea has been around as long as man has been able to move across water. The first recorded amphibious assault took place in 490 B.C. when the Persians attacked the Greeks at Marathon [RODG37]. Typically, specialized equipment was not used because opponents had to fight hand-to-hand anyway. Fighting at sea, on the beaches, or on land was essentially the same. The only difference was whether or not you would drown if knocked out. When Kublai Khan sent the Mongol hordes against Japan in 1274 and 1281, they had specialized landing boats and actually attacked against defended beaches [MARD83]. In 1801, the British at Aboukir Bay in Egypt used specialized flat-bottom landing craft [RYAN83], as did the Americans in 1849 at Vera Cruz in the Mexican War [BAUE69].

With the advent of longer range and more accurate weapons, the tricky business of amphibious assault across defended beaches became even more difficult. Already one of the most problematic logistical headaches, the better weapons could turn an assault into a turkey shoot as unprotected landing craft move across the open water and open beaches. One possible solution was hit upon by a man named Roebling in 1932. His idea was an aluminum hulled amphibious vehicle which used tracks for propulsion in water and on land. In 1939, after 7 years of development, the Roebling Alligator could do 25 mph on land and 8.6 mph in the water. In 1940, the next generation, the Roebling Crocodile, was copied directly (except using sheet steel instead of aluminum) and became the LVT-1 for military use. The vehicles have been in use since then and have been through many generations up to the current version, the AAV7A1. [AASB94]

The latest AAV can move at 45 m.p.h. on land and 8.2 m.p.h. in the water. It can go over an 8 foot trench, a 3 foot vertical wall, and climb a 60% grade slope. The AAV7A1

can safely negotiate a 6 foot plunging surf while fully loaded. It can survive a 10 foot plunging surf with little damage and has survived (in tests) a 20 foot plunging surf. The aluminum armor is sufficient for small arms, light machine guns, and artillery shell bursts. The AAVP7A1 (P for passenger) has a turret containing a M2 .50 Caliber Heavy machine gun and a Mk19 40 mm automatic grenade launcher. Its main weapon is the contingent of 21 fully combat loaded Marines it can hold in addition to its three crewmen. Once on land, the vehicle functions as an armored personnel carrier, able to project force deep ashore. [TECH93]

C. APPROACH

One of the basic premises of ModSAF's design is to allow the addition of new vehicle types, units, formations, and weapons without having to recompile the program. All of the previous items are created at run time using "reader" files. A "reader" file is a text file containing the necessary information in a particular format. To create an AAV from the ground up would be a daunting task. Specifying all the necessary parameters for abilities, behaviors, and weapons could take months of work. Whenever creating a new type of vehicle, the creators of ModSAF recommend that you find a similar vehicle, copy it, and make the necessary changes [MODS95]. Fortunately for the author, a similar vehicle does exist, the M2 Bradley Fighting vehicle.

The approach taken was to create an AAV using as much as possible from the M2 Bradley. Additionally, to create the amphibious assault mission discussed in Chapter V, a ship was also created. ModSAF already had a Dismounted Infantryman (DI) modeled. The most difficult part was adding the behaviors to enable the DIs to mount onto the AAVs, and the AAVs to embark upon the ship. The dismounting/disembarking behavior was also added along with an amphibious assault mission.

The decision to base the new AAV entity upon the already modeled M2 proved to be a good one. Although lacking the amphibious capability, the similarities in behavior and

armament were quite helpful. As discussed below, the change from a strictly land vehicle to an amphibious one turned out to be fairly simple (if you know where to look).

D. IMPLEMENTATION

1. AAV

In the ModSAF Software Architecture and Design Overview Document (SADOD), there is a chapter on how to extend the program for new vehicle types [MODS95]. The document lists the seven necessary steps, as in Figure 4, and then explains each step in detail. All the changes made to each file to create the AAV are listed in Appendix A. Due to the uniqueness of the vehicle, several new object domains, environments, and vehicle types were added to the protocol. The key part of the changes, where the vehicle is defined to be amphibious, took place in the parameter file. This is step 2 of Figure 4.

1. Define the protocol constants of the new vehicle
2. Create or modify the vehicle parameter file
3. Load the parameter file
4. Reference the new parameters
5. Add the vehicle to the graphical user interface
6. Add vehicle and military icons
7. Check the physical database

Figure 4: Seven Steps to Create a Vehicle in ModSAF

In the parameter file, US_AAVP7_parameters.rdr, the default parameters for the process that controls the vehicle movement according to terrain (SM_VTerrain) were over-ridden to not avoid deep water. This one override enabled the vehicle to move across deep water. The overrides also changed the maximum speed of the vehicle while it is in water.

New symbols were created to represent the individual vehicles and units. These new symbols are shown in Figure 5.

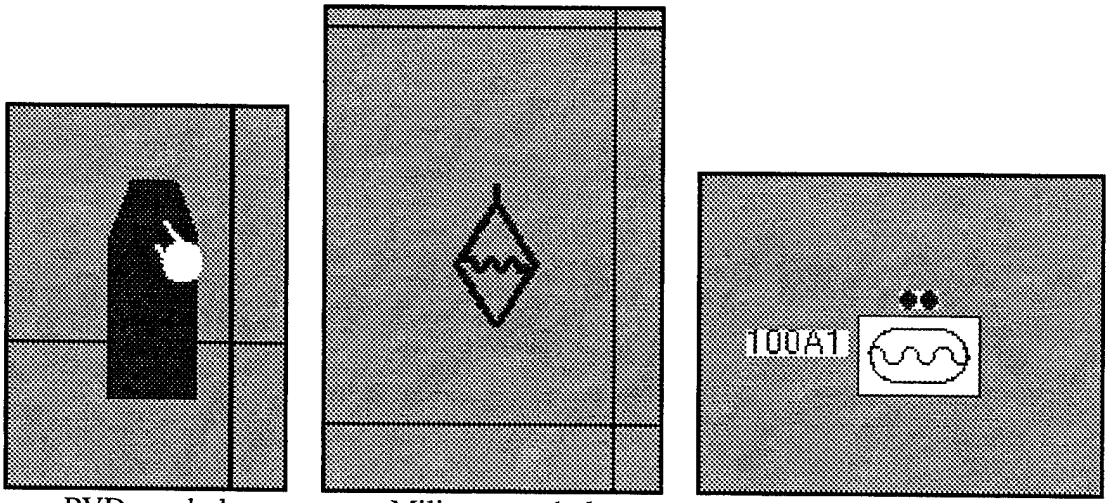

Figure 5: New 2D AAV Symbols

2. LHA

The LHA (Landing Helicopter Assault) amphibious ship created for use in the amphibious assault mission has very minimal abilities. Appendix B lists how it was created. The only behavior (other than defaults) that it has is the ability to move. This is sufficient for the purposes of this study. As will be discussed in the next chapter, ModSAF does not have the ability to embark vehicles onto a ship. This behavior was added specifically for this study.

E. LIMITATIONS

In the current version of ModSAF (version 1.5.1), there is no way to account for sea-state. Vehicles on the ground will slow down or speed up according to the roughness of the terrain. However, vehicles in the water will move at their maximum limits specified for deep water without regard to how rough the sea is.

The carrying of troops and the embarkation of the vehicle upon another vehicle, such as the ship in Figure 6, is not currently intrinsic to the vehicle itself. The behavior is simulated by making the appropriate vehicle or troop disappear and reappear at the right

time. A programmer could "load" 1000 troops onto the vehicle without the lower, physically-based, functions protesting.

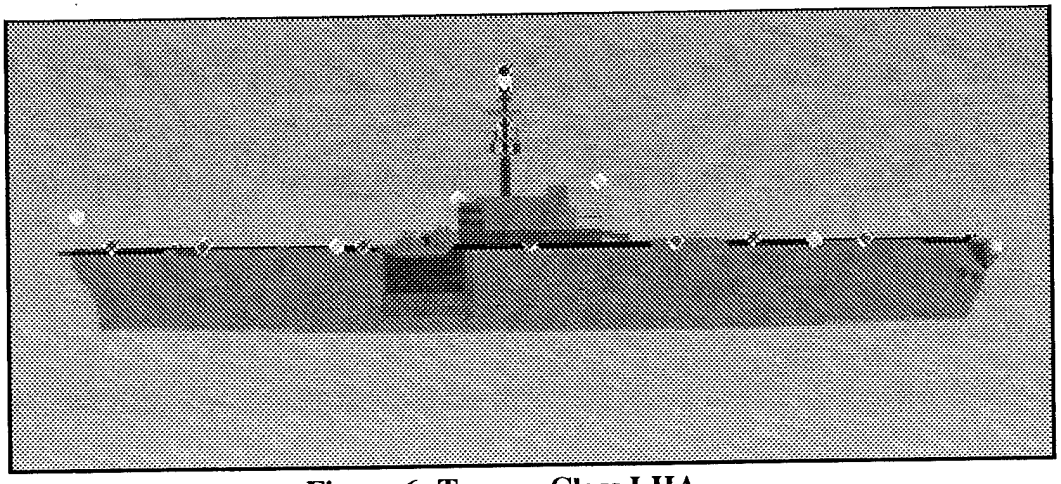

Figure 6: Tarawa Class LHA

F. SUMMARY

The resulting vehicle, an AAVP7A1, moves and shoots in a physically-based manner on land and on water. The loading of troops onto the AAV, and the AAV onto a ship, is a simulation work-around that is not physically-based. The ship created, a Tarawa Class LHA, has very limited abilities and is not a model for emulation. It serves the function of transporting AAVs and DIs to the debarkation point for the amphibious assault mission.

V. AMPHIBIOUS ASSAULT MISSION

"Ever since the days of the Phoenicians, the ability to land on defended shores has been a source of strength for those who possess it and a source of concern for those who must oppose it." [BARR83]

A. OVERVIEW

Good seaports and trading routes have been the hallmarks of powerful countries for thousands of years. Most countries have some coastline or are accessible from water routes which make them vulnerable to an amphibious assault. The ability to conduct an assault against defended beaches gives the aggressor a much larger range of options than would be available otherwise. Since before World War II, the United States has made the commitment to have those options available if needed.

The classic amphibious assault consists of "waves" of assault craft moving against a dug-in enemy. The difficulty for the attackers is to move enough power ashore quickly to carve out a toehold from which to pour in even more forces. The difficulty for the defenders is that an amphibious assault can be made from anywhere along a coastline. The sheer amount of defensible area makes it difficult to reinforce all of it. Unquestionably, the most dangerous part for the attackers is during the ship-to-shore movement. Closely timed naval bombardment that ceases just as the attackers hit the beach is key, as is minimizing the time involved in moving to shore.

Advances in technology have complicated the entire process. There are a large variety of relatively cheap missiles that can take out a ship. As a result, naval bombardment becomes more difficult and the navy prefers to stay "over-the-horizon". It is now easier than ever to obtain sophisticated mines to sprinkle along the coastline. But one thing is clear, the need for an amphibious assault capability will be around a long time. Therefore, a need exists to be able to simulate the process.

B. MISSION SELECTION

1. Types of Amphibious Operations

There are four types of amphibious operations. First is the amphibious assault which puts a military force upon a hostile shore. Second is the amphibious demonstration which is a fake amphibious assault to draw enemy forces into defending the shoreline. Third is the amphibious raid in which a force goes ashore, accomplishes some mission, and then returns. The fourth type is an amphibious withdrawal in which forces are pulled back from a hostile shore. Any of the above missions would have satisfied the requirement for proof-of-concept of the AAV.

2. Simplified Mission

A real amphibious assault would consist of much more than AAV's, carrying troops and launched from an LHA, hitting the beach. Preparations would begin at least six months in advance. Detailed plans and practice assaults would take place. Every detail for every contingency would be discussed and written up. All the supplies needed for three to thirty days would loaded on the ships. Loading would be carefully planned so that ammunition, fuel, and food would be off-loaded before any non-critical items. The assault itself must be coordinated with naval bombardment, fixed wing support, helo support, and "vertical envelopment" forces (i.e. troops brought in by helo's and landing behind the enemy). Navy SEAL and recon forces must move in prior to the assault to clear any mines or obstacles. Once an assault is set in motion, there would be greater casualties from stopping than from carrying on against almost any defense.

The intent of this study was not to fully simulate the complete amphibious assault. A greatly simplified amphibious assault mission was designed and performed the following steps:

1.) Created a unit consisting of one LHA, a section of AAV's and a platoon of DI's (Dismounted Infantry) using the "unit editor". Position it as shown in Figure 7.

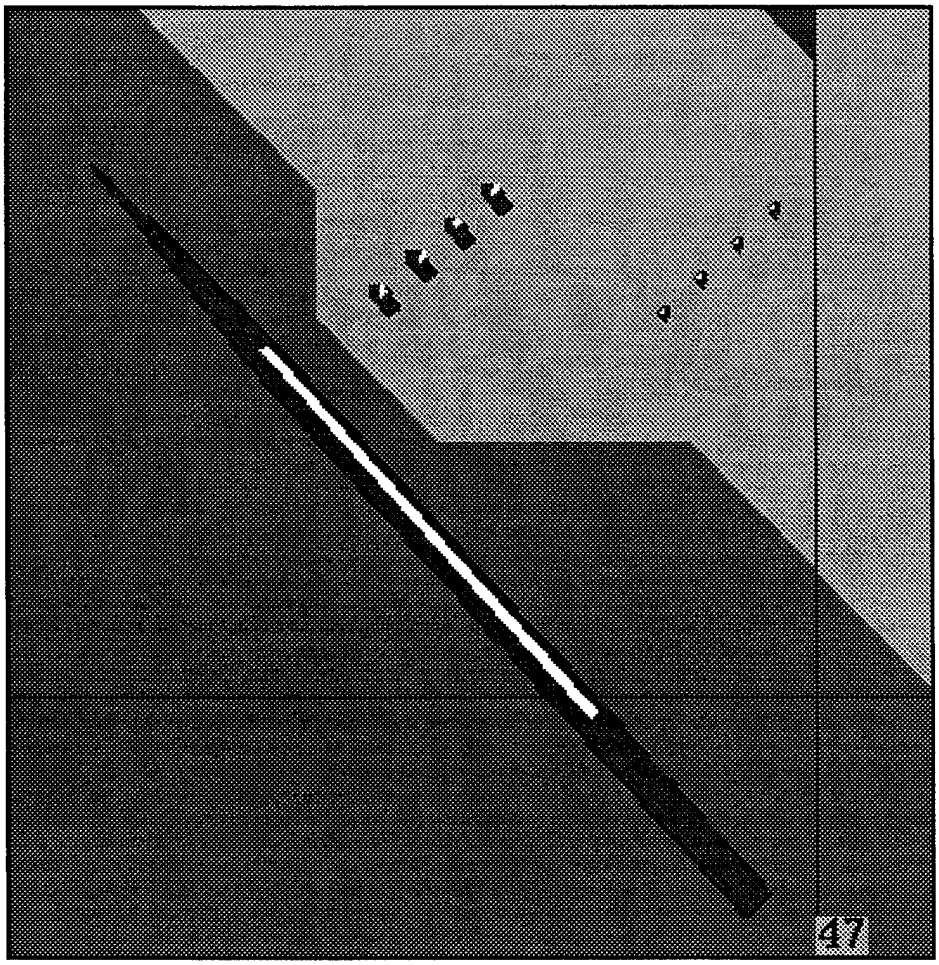

Figure 7: LHA / AAV / DI Unit

2.) Selected the "Embark" task and the AAVs picked up the DIs and embarked onto the LHA.

3.) The LHA was then moved to a position off the target beach using "Move".

4.) The amphibious assault task was then selected. The AAVs disembarked, moved to shore in a "wave", and proceeded to occupy the designated beach. Once in position onshore, the DIs disembarked the AAVs.

C. APPROACH

The simplified mission above required the construction of five separate finite state machines. Version 1.5.1 of ModSAF does not have the ability to embark or disembark vehicles from ships. There is a limited ability to mount and dismount DI's from IFV's (Infantry Fighting Vehicles), but this has been specifically limited to one kind of unit.

This study has also limited the embark / disembark / amphibious assault ability to one very specific unit. To write a general solution would be better in the long run but far more time consuming in the short run. The unit capable of this mission is an LHA/AAV/DI unit. It consists of one LHA, four AAVs (a section), and four DI's (which represent a platoon).

1. FINITE STATE MACHINES

The five AASFMs (Augmented Asynchronous Finite State Machines) created were all modeled after existing libraries. The machine that handles the embarking and disembarking at the individual vehicle level, libvembark, closely resembles libvmount. The machines that handle the unit level embarking and disembarking, libuebkpckup and libudisembk, closely resemble libupickup and libudsmnt, respectively. And at the top of the hierarchy, the machines that handle mixed (i.e. multiple) units embarking and disembarking, libumxembark and libumxdisembk, are similar to libumxmount and libumxdsmnt, respectively.

As shown in Figure 8, each level calls on the next lower level until libvembark is reached which actually controls the individual vehicle embarking or disembarking. In reality, libvembark also calls upon libvmove to actually move the vehicles around. This type of hierarchical control is typical of ModSAF. Each AAFSM that controls larger units, spawns tasks by passing portions of that unit onto other AAFSMs until the individual vehicle is finally acted upon. Note that the "Mixed Unit Embark" can be called from the mission editor, but the "Mixed Unit Disembark" must be called from another task. This design was intentional and could be changed with relative ease.

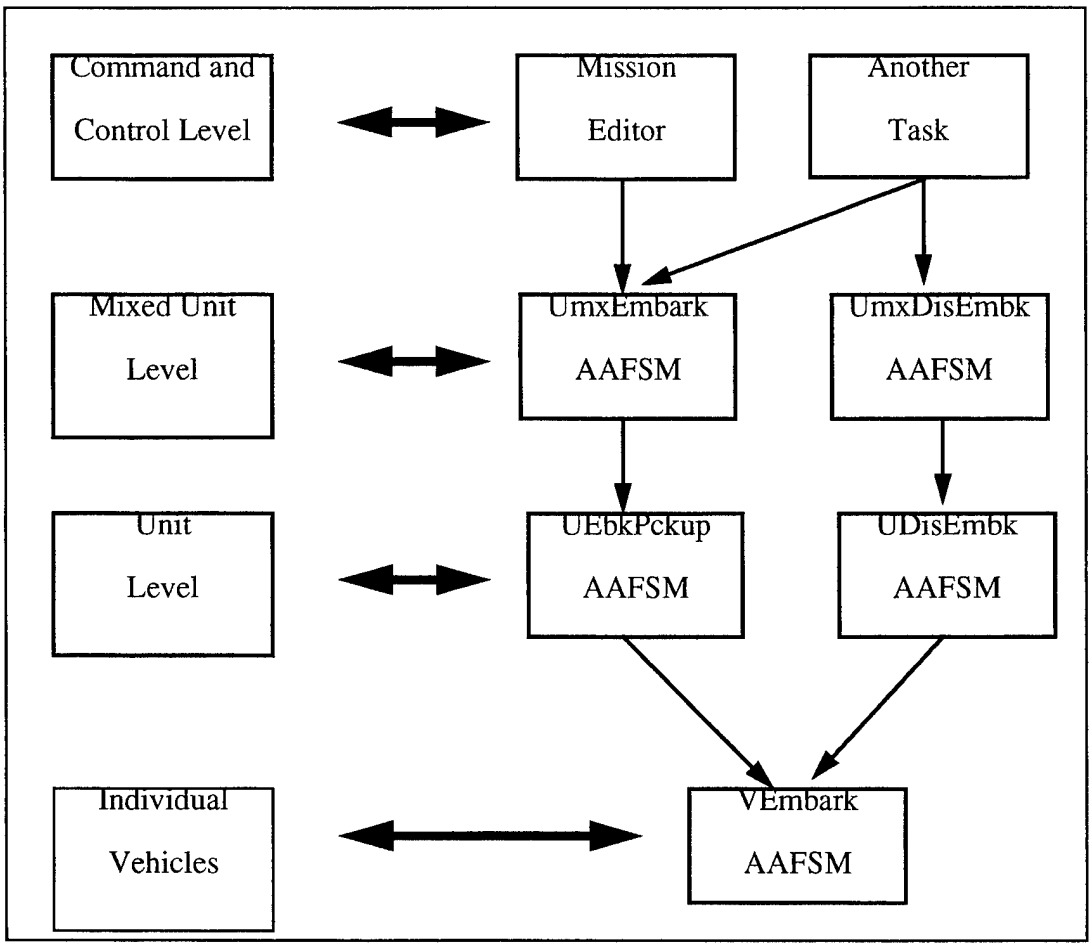

Figure 8: Finite State Machine Hierarchy

D. IMPLEMENTATION LIMITATIONS

As stated previously, the amphibious assault mission is very specific to one particular unit. The behaviors necessary to implement the mission are only available to the LHA/ AAV/DI unit. It is the only unit that can mount/dismount DIs on AAVs and embark/ disembark AAVs from an LHA. This was done to simplify the coding while still demonstrating the proof-of-concept that military operations in littoral regions can be simulated.

VI. FUTURE VEHICLE

A. OVERVIEW

The current version of amphibious assault vehicle has been in service since 1972. Its replacement is not scheduled for full production until 2004. One candidate for the job is shown in Figure 9 below. The new vehicle must be able to keep up with the M1 Abrahms tank (60 m.p.h. +) and move over the water three times faster than the current version. All the possible replacements use some type of water-jet propulsion and movable planes to allow the vehicle to ride like a speedboat.

The new vehicle allows changes in tactics to respond to the greatly improved weapons available on the world market. Increased maneuverability will open up larger potential areas of attack while keeping the expensive navy ships at a safe distance. How to make best use of the new capabilities is a perfect opportunity for modeling and simulation.

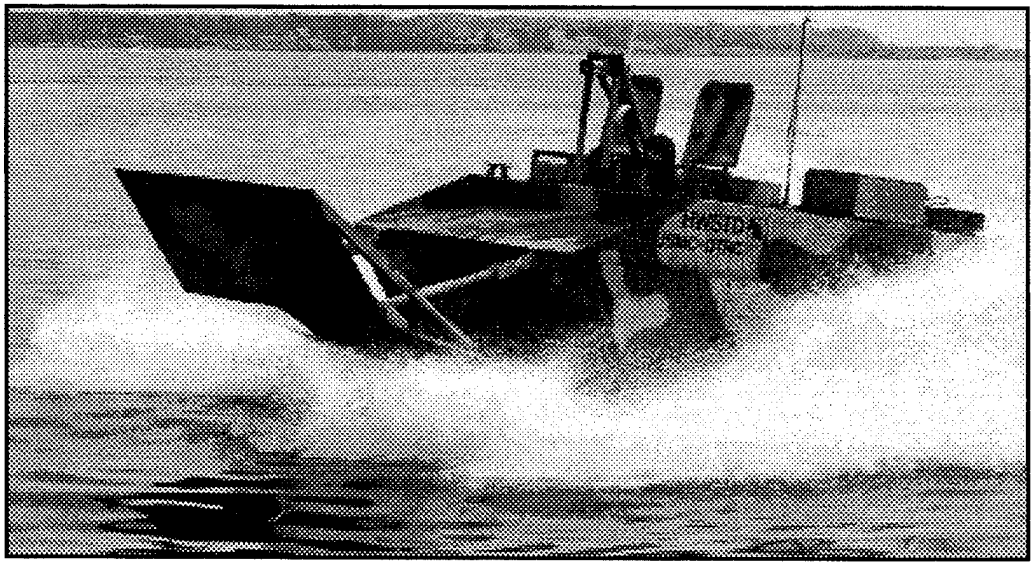

Figure 9: Advanced AAV Prototype

B. TACTICAL ADJUSTMENT

The main problem that the new vehicle must solve is simple: speed. The current AAV cannot keep up with the M1 Abrahms tank on land and cannot be launched too far from shore. This problem is what led to the development of the hovercraft discussed later. If the AAAV is able to meet the specifications, tactical changes are imminent.

1. AAAV

Current AAV amphibious assault tactics call for launching the vehicles about 2500 yards from the target beach. This puts the ship doing the launching within range of many missiles, artillery, and small attack boats. The further the ship moves out, the longer the AAV's must bob around the ocean, reducing the combat effectiveness of the Marines inside. Additionally, the slow moving AAVs spend more time as easy targets for land-based weapons.

With the creation of the AAAV, many old tactics and paradigms must be thrown out. The time spent on the water, within range of land-based weapons, will be greatly reduced. The ride, while still very rough, will not cause as much seasickness. The launching ships can remain over-the-horizon and have more time to respond to longer range threats.

One particular problem is the lack of naval bombardment support available since the ships will be staying so far out. The retirement of every battleship, without replacement, has also compounded the problem. The new AAAV will have a stabilized weapon turret. This will make the vehicle far more effective on land as a base-of-fire and enable it to engage targets accurately during an amphibious assault. This is a partial solution to a very difficult problem.

2. LCAC (Hovercraft)

One of the driving forces in the development of the LCAC (Landing Craft, Air Cushion) or hovercraft was the speed at which it could move over the water and over land. The LCAC can go over 60 knots on the water and, theoretically, the same over land. This allows the vehicle to be launched from long distances and still achieve surprise at the

appropriate land fall. Unfortunately, the vehicle cannot support the weight of armor and does not have any armament. It is actually relatively fragile and certainly not ideal for assaulting a defended beach. Even with these disadvantages, this impressive machine can be extremely useful for a variety of missions.

One possibility is the rapid movement ashore of heavy weaponry. Once the initial toehold on a beach is established, the LCAC could carry in larger artillery pieces and more ammunition. This would give the initial forces a much stronger position. The rapid replacement of supplies is essential to an assault. Initial forces must go in lightly weighted to allow maximum maneuverability. With its speed, the LCAC would be able to bring in timely supplies. There have also been experiments with the LCAC carrying troops inside containers and carrying the current AAV.

C. SIMULATION REQUIREMENTS

To add the new Advanced AAV to future versions of ModSAF should be straightforward. Once the new vehicle is identified as a new model of the current AAV, simple changes to the copied parameter file will give it all the capabilities of the old AAV with new characteristics. This is a simplification, but adding a new version of an existing vehicle is not difficult.

To add the LCAC would be more difficult but not dramatically so. The current LCAC is more comparable to a very fast supply ship with limited amphibious capability. The LCAC does not perform particularly well on land due to the amount of dust and objects kicked up by its powerful fans. It is sufficient to move over smooth surfaces, but not suited to moving over rough terrain.

VII. SUMMARY AND CONCLUSIONS

A. SUMMARY

The intent of this study was to demonstrate that an entity could be created in ModSAF that is capable of moving cross paradigms, and therefore capable of simulating amphibious assault operations. The entity created, the Assault Amphibious Vehicle, must be able to operate in water and on land in the same manner the real AAV does. This provided the crucial link in demonstrating an amphibious assault. Now land entities (Dismounted Infantryman) and sea entities (Amphibious Warfare ships) can utilize the AAV entity to conduct assaults from the sea.

B. RESULTS OF WORK

1. Assault Amphibious Vehicle

The addition of a new vehicle type, capable of crossing from land to water and back, turned out to be easier than expected. This was mainly due to the extensible design of ModSAF. Most of the existing work in ModSAF has concentrated on land vehicles. As a result of the extensible design, the new vehicle was able to subsume the behaviors of the already existing land vehicles. The modification to allow the vehicle to move across deep water turned out to be a small change in the parameter file. The program does not take into account the relative roughness of the water like it accounts for the roughness of the terrain. The vehicle simply moves much slower in water than on land. Additionally, or as a side effect, there is no accounting for behavior in the surf zone. The transition point from sea to beach is typically the most dangerous as waves can actually lift up and swamp the 25-ton craft. Otherwise, the physically based behavior of the vehicle is accurate.

2. Amphibious Assault Mission

The amphibious assault mission was intended as a demonstration of the capabilities of the new AAV. Unfortunately, version 1.5.1 of ModSAF did not have the necessary behaviors to conduct a limited assault from a ship. The ability to mount DIs into an AAV and the ability to embark the AAV on a ship needed to be added. Disembarking and dismounting also needed to be added. Additionally, there were no ships available even though the protocol had definitions for them. Other behaviors, such as assaulting a position and occupying it, were already present.

The resulting mission is very specific and only works for one particular unit. An LHA/ AAV/DI unit can embark, move to position, disembark, and assault a beach. The behaviors have very few options and are very specific.

C. CONCLUSIONS

Amphibious military operations in littoral regions can be simulated by extending the current programs used to generate SAF. The extension will not require any major shifts in architecture or design paradigms. In particular, the ModSAF program was able to handle the changes with relative ease.

D. RECOMMENDATIONS FOR FUTURE WORK

As work continues on extending ModSAF to handle Navy and Marine Corps specific needs, many behaviors currently unavailable must be added. The most obvious behavior needed is a generalized ability for DIs to mount and dismount from any vehicle. Along a similar line, the ability for vehicles to hold other vehicles must be added. Current behaviors are very limited and specific to certain units.

The need to simulate the Advanced AAV will be necessary soon for the development of tactics and methods of employment. Simulation would help try new sizes of units and various restructuring of organizations. By the time the real one is available, the Marines should be ready to deploy it from the factory.

APPENDIX A: AAV VEHICLE

In the SADOD document [MODS95], the seven steps necessary to create a new vehicle are listed and described in detail. The changes made to each file to create the AAV are described below.

A. PROTOCOL CONSTANTS

All the following files reside in 'common/include/protocol/'. Anytime changes are made to these files, a 'gmake' must be done in the directory 'common/libsrc/librdrconst' to install the new protocol definitions.

'veh_type.h':

```
/* Amphibious Assault Vehicles */
#define SP_vehicle_US_AAVP7          \
     ( SP_objectDomainVehicle | SP_vehicleEnvironmentAmphibious \
      | SP_vehicleClassAmphibious | SP_vehicleCountryUS \
      | 0 << SP_vehicleSeriesShift | 0 << SP_vehicleModelShift \
      | SP_vehicleFunctionAmphibiousAPC )
```

'p_safmodels.h'

```
#define SP_SM_AmphibiousHull  ( SP_SM_classHull | 8 << SP_SM_instanceShift )
```

'obj_type.h'

```
#define SP_vehicleEnvironmentAmphibious    ( 5 << SP_vehicleEnvironmentShift )
#define SP_vehicleClassAmphibious          ( 6 << SP_vehicleClassShift )
#define SP_vehicleFunctionAmphibiousAPC    18 /* AAV */
#define SP_echelonEnvironmentAmphibious    ( 5 << SP_echelonEnvironmentShift )
#define SP_echelonClassAmphibious          ( 6 << SP_echelonClassShift )
#define SP_echelonFunctionAmphibiousAPC    19 /* AAV formations */
```

In addition to the above changes, similar definitions were made to create units of AAV's.

B. PARAMETER FILE

In the individual vehicle parameter file, any parameters that needed to be different than those contained in the 'standard_params.rdr' or 'macros.rdr' were over-ridden at this point. The comments at the beginning of the file provide a good explaination of the 'inheritance' method used to define vehicle behaviors and abilities. These files are in the directory 'common/src/ModSAF/entities/':

'US_AAVP7_params.rdr':

```
;; $Revision: 1.0 $
;;
;; This file defines all the model parameters for a vehicle_US_AAVP7

;; Note:  There are currently three levels of specificity in the vehicle
;;        .rdr files.  They are, from the general to the specific:
;;
;;        - standard_params.rdr
;;        - macros.rdr
;;        - the individual vehicle .rdr files
;;
;;        When adding a parameter, you must determine in which of the above
;;        three files it belongs.  Generally, this set of rules may be
;;        followed:
;;
;;        - If the parameter has the same value(s) for all vehicles of a
;;          category (the categories being ground vehicles, fixed wing
;;          aircraft, rotary wing aircraft, and missiles), then it belongs
;;          in standard_params.rdr
;;
;;        - If the parameter has the same value(s) for many vehicles, but
;;          not all vehicles of a category, then make a macro out of it,
;;          put the macro in macros.rdr, and refer to the macro in the
;;          appropriate vehicle .rdr files
;;
;;        - If the parameter is different for most vehicles, then it is
;;          best to specify it in each vehicle .rdr file
;;
;;        When looking for a parameter, start at the individual vehicle .rdr
;;        file.  If it's not there, look in macros.rdr.  If it's not there,
;;        look in standard_params.rdr.


US_AAVP7_MODEL_PARAMETERS {

(SM_Entity DEFAULT_DEAD_RECKONING_PARAMETERS
        (vehicle_class vehicleClassAmphibious)
        (guises vehicle_US_AAVP7 vehicle_USSR_BMP)
```

```
        (send_dis_deactivate true)
        )

;; Remove ;;'s to turn reaction to stingray fire on
(SM_DFDamage (filename "dfdam_IFV.rdr")
            (damage_threshold 10.0)
;;          (stingray
;;           (damage_table_laser1 "dfdam_sr_invincible.rdr")
;;           (damage_table_laser2 "dfdam_sr_invincible.rdr")
;;           (damage_duration 15.0 25.0)
;;           (sight_occupancy 0.8)
;;          )
)

(SM_IFDamage (name apc1))

;; Remove ;;'s to turn reaction to stingray fire on
;;(SM_SRDetect (filename "srdet_table_vulnerable.rdr"))
;;VSRREACT

(SM_Components (hull SM_TrackedHull SAFCapabilityMobility)
          (primary-turret [SM_GenericTurret | 0])
          (commander-sight [SM_Visual | 0])
          (driver-sight [SM_Visual | 1])
          (gunner-sight [SM_Visual | 2])
          (main-gun [SM_BallisticGun | 0] SAFCapabilityFirepower)
          (machine-gun [SM_BallisticGun | 1] SAFCapabilityFirepower)
          )

(SM_TrackedHull (soils (SOIL_DEFAULT      (max_speeds 65.18 65.18) ;40.5 MPH
                                          (max_decel 4.98) ; 14 fss
                                          (max_climb 60.0)
                       SOIL_DEFAULT_TRACKED)
                (SOIL_ROAD          (max_speeds 65.18 65.18)
                                          (max_decel 4.98)
                                          (max_climb 60.0)
                       SOIL_ROAD_TRACKED)
                (SOIL_RCI250        (max_speeds 65.18 65.18)
                                          (max_decel 4.98)
                                          (max_climb 60.0)
                       SOIL_RCI250_TRACKED)
                (SOIL_RCI050        (max_speeds 43.45 43.45) ; 27 MPH
                                          (max_decel 4.98)
                                          (max_climb 60.0)
                       SOIL_RCI050_TRACKED)
                   (SOIL_SHALLOW_WATER  (max_speeds 21.73 21.73 ) ;13.5
MPH
                                          (max_decel 8.0)
                                          (max_climb 60.0)
                       SOIL_SHALLOW_WATER_TRACKED)
                (SOIL_DEEP_WATER    (max_speeds 13.0 13.0) ; 8 MPH
```

```
                                        (max_accel 0.44)
                                        (max_decel 10.0)
                                        (max_climb 60.0)
                                SOIL_DEEP_WATER_TRACKED)
                )

            (fuel_usage (0.0   100.0)
                    (0.125  12.5))
            )

;; Primary turret
([SM_GenericTurret | 0] (physdb_name "primary-turret")
                (rates continuous 0.0 40.0))

;; Main gun - MK19 Mod 3 40 mm machine gun
([SM_BallisticGun | 0]  (physdb_name "main-gun")
                (sensor_name "gunner-sight")
                (hit_obscuring_vehicles true)
                (rates 0.0 40.0)
                (magazine_size 96)
                (loading_block 96)
                LOAD_TIME_MACHINE_GUN
                (munitions (munition_US_M430
                        (round_velocity 242.0)
                        (rate 375)
                        (mass 0.4)
                        (min_range 18.0)
                        (max_range 2212.0)
            HIT_TABLE_SIMPLE_NO_LASER_KE
            TRACKTIME_TABLE_MACHINE_GUN)
                        )
                )

;; Machine  gun - Browning .50 (12.7mm) HB, M2
([SM_BallisticGun | 1]  (physdb_name "machine-gun")
                (sensor_name "gunner-sight")
                (hit_obscuring_vehicles true)
                (rates 0.0 40.0)
                (magazine_size 200)
                (loading_block 200)
                LOAD_TIME_MACHINE_GUN
                (munitions (munition_US_M33 ;; .50 cal. ball
                        (round_velocity 934.0)
                        (rate 550)
                        (mass 0.12)
                        (min_range 0.0)
                        (max_range 6700.0) ;
            HIT_TABLE_SIMPLE_NO_LASER_HE
            TRACKTIME_TABLE_MACHINE_GUN)
                        )
                )
```

```
;; Commander's sight
([SM_Visual | 0] VISUAL_APC_COMMANDER_DVO )

;; Driver's sight
([SM_Visual | 1]VISUAL_APC_DRIVER_DVO )

;; Gunner's sight
([SM_Visual | 2]VISUAL_APC_GUNNER_DVO )

(SM_Supplies (munition_Fuel 649.8);; 171 gallons, in liters.
        (munition_US_M430 864.0);; 40mm HEDP
        (munition_US_M33 1200.0) ;; .50 cal. ball
        )

(SM_VSpotter (background on)
        (sensors commander-sight driver-sight gunner-sight)
        VSPOTTER_SPECS
        )

(SM_VTargeter
 (background on)
 (munition_control_types
  (munition_US_M430
   (type ballistic)
   (tracking_data (name          main-gun)
          (is_gun          true)))
  (munition_US_M33
   (type ballistic)
   (tracking_data (name          machine-gun)
          (is_gun          true)))
 )
 (default_weapon "main-gun" munition_US_M430)
 VTARGETER_GROUND_TRACKED
 )
(SM_VAssess (background on)
        (sensors commander-sight
                driver-sight
                gunner-sight)
        (weapons ([objectDomainMask     |
                vehicleEnvironmentMask |
                vehicleClassMask     |
                vehicleFunctionMask   ]
                [objectDomainVehicle       |
                vehicleEnvironmentGround    |
                vehicleClassSPArmoredTracked |
                vehicleFunctionMainBattleTank]
                (0.0 3500.0 "main-gun" munition_US_M430)
                )
                ([objectDomainMask     |
                vehicleEnvironmentMask ]
```

```
            [objectDomainVehicle      |
             vehicleEnvironmentGround]
            (0.0 3500.0 "main-gun" munition_US_M430)
            )
           ([objectDomainMask     |
             vehicleEnvironmentMask |
             vehicleClassMask     ]
            [objectDomainVehicle   |
             vehicleEnvironmentAir |
             vehicleClassRotaryWing ]
            (0.0 3500.0 "main-gun" munition_US_M430)
            )
           (objectDomainMask
            objectDomainLifeForm
            (0.0 2000.0 "machine-gun" munition_US_M33)
          )
           )
       (max_hits_on_target 20)
       (gunner_visual "gunner-sight")
       VASSESS_GROUND
        VASSESS_IFV_THREATS
        VASSESS_IFV_OPTIONAL
       (no_target_load
         ("main-gun" munition_US_M430)
         ("machine-gun" munition_US_M33)
        )
       )

(SM_VTerrain
 (entity_period 1000)
 (avoidance_mask [
            VTERRAIN_BUILDING |
;           VTERRAIN_WATER |
;           VTERRAIN_TREELINE |
;           VTERRAIN_TREE |
            VTERRAIN_CANOPY |
            VTERRAIN_ENTITY |
          VTERRAIN_STEEP_AREA |
            VTERRAIN_BREACH_LANE
            ])
 (avoid_soils SOIL_NO_GO)
 (background on)
 (movement_threshold 500.0)
 (map_radius 1000.0)
 (entity_radius 500.0)
 (history_list_spacing 20.0)
 (num_history_list_points 50)
 (breach_obst_nominal_size 400.0))

}
```

C. LOAD PARAMETER FILE

The newly created parameter file was added to 'modellist.rdr' which is located in 'common/src/ModSAF/entities/':

"US_AAVP7_params.rdr"

D. REFERENCE PARAMETER MACRO

The individual vehicles use other macro definitions to handle any parameter values not specified in the vehicle parameter file. These macros are associated with the vehicle in the file 'models.rdr'. This file is also located in 'common/src/ModSAF/entities/':

("vehicle_US_AAVP7" US_AAVP7_MODEL_PARAMETERS
GROUND_STD_PARAMS)

E. ADD TO GUI

The graphical user interface has the available vehicles input during start-up. By adding the vehicle to 'libsrc/libunits/units.rdr' and doing a gmake, the vehicle will appear on the list of available entities:

("AAVP7" vehicle_US_AAVP7)

F. ADD ICONS

There are several display options for vehicles and units. They can be shown as PVD (Plan View Display) pictures, military individual icons, military platoon icons, or military company icons. Each level has a defaulting behavior that will bring up an icon based on the vehicle role. To define a unique PVD, it was necessary to edit the file 'pvd.rdr' in the directory 'common/libsrc/libpvd' and run 'gmake':

```
PVD_AAV_HULL{ PVD_HULL
                (block-0.5 0.25
                    -0.25 0.5
                     0.25 0.5
                     0.5 0.25
                     0.5 -0.5
                    -0.5 -0.5)
              }
PVD_AAV_HTURRET{ PVD_TURRET
                (disc0.0 0.0 0.5 x)
                (line0.25  -0.25 -0.3 -0.25 1.0)
```

```
                (line0.25  0.25 -0.3 0.25 1.5)
               }

PVD_AAV_FTURRET{ PVD_TURRET
                (disc0.0 0.0 0.5 x)
                (line0.25  -0.25 -0.3 -0.5 0.5 -0.25 1.0)
                (line0.25  0.25 -0.3 0.0 0.5 0.25 1.5)
               }

PVD_AAV_PICTURES{ (;; Healthy
                ((PVD_TEAMPVD_BLACKPVD_AAV_HULL)
                 (PVD_WHITEPVD_NONEPVD_AAV_HTURRET))
                ;; MKill
                ((PVD_TEAMPVD_BLACKPVD_MKILL_HULL)
                 (PVD_WHITEPVD_NONEPVD_AAV_HTURRET))
                ;; FKill
                ((PVD_TEAMPVD_BLACKPVD_AAV_HULL)
                 (PVD_WHITEPVD_NONEPVD_AAV_FTURRET))
                ;; MFKill
                ((PVD_TEAMPVD_BLACKPVD_MKILL_HULL)
                 (PVD_WHITEPVD_NONEPVD_AAV_FTURRET))
                ;; KKill
                ((PVD_BLACKPVD_TEAMPVD_MKILL_HULL)
                 (PVD_BLACKPVD_TEAMPVD_AAV_FTURRET)))
               }
;; Representative amphibious armored personnel carriers
(vehicle_US_AAVP7 PVD_AAV_PICTURES)
```

To specify a correct military icon for the AAV, it was necessary to edit the file 'bgrdb.rdr' in the directory 'common/libsrc/libbgrdb/' and run 'gmake':

```
PVD_US_AAV_ICON { (PVD_ROTATE
            (PVD_DEFAULT
             ("elt Diamond /Hold"
            "elt RAmphibious"
              "elt UCompany")))
            }
;; Specific amphibious assault vehicles
  (vehicle_US_AAVP7  PVD_US_AAV_ICON)
```

G. PHYSICAL DATABASE

The physical database is where the actual physical characteristics of the vehicle are input. The file 'physdb.rdr' is contained in the directory 'common/libsrc/libphysdb/' and a 'gmake' is required for changes to take effect:

```
(vehicle_US_AAVP7
 (3.33 8.2 3.33 0.0 0.0 25792.0
  1830.0 ; max effective range of .50 cal. - range of Mk 19 is 1500 m
  OPTICAL_CONTRAST_MACRO
  IMAGEI_CONTRAST_MACRO
  THERMAL_CONTRAST_MACRO
  ((primary-turret 0 true 1.06 1.06 2.5 1.80 2.30 0.80 360.0 360.0
    ((main-gun 1 true -0.25 0.5 0.4 0.0 1.1 0.0 0.0 45.0 -8.0 0)
     (machine-gun 2 true 0.25 0.5 0.4 0.0 1.66 0.0 0.0 45.0 -8.0 0))))))
```

APPENDIX B: LHA VEHICLE

In the SADOD document [MODS95], the seven steps necessary to create a new vehicle are listed and described in detail. The changes made to each file to create the LHA are described below.

A. PROTOCOL CONSTANTS

All the following files reside in 'common/include/protocol/'. Anytime changes are made to these files, a 'gmake' must be done in the directory 'common/libsrc/librdrconst' to install the new protocol definitions. Many protocol definitions were already in place for an LHA vehicle even though such a vehicle was not available for use.

'veh_type.h':

```
  /* LHA 1 - USS Tawara assault */
#define SP_vehicle_US_LHA1          \
    ( SP_objectDomainVehicle | SP_vehicleEnvironmentWater \
    | SP_vehicleClassAmphibWarfare | SP_vehicleCountryUS \
    | SP_USSTawaraClass | 1 << SP_vehicleModelShift \
    | SP_vehicleFunctionAmphibAssault )/* Amphibious Assault Vehicles */
```

'obj_type.h'

```
#define  SP_vehicleEnvironmentWater          ( 4 << SP_vehicleEnvironmentShift )
#define SP_vehicleClassAmphibWarfare        ( 1 << SP_vehicleClassShift )
#define SP_vehicleFunctionAmphibAssault       2
#define SP_echelonFunctionAmphibAssault       2
#define SP_echelonEnvironmentWater         ( 4 << SP_echelonEnvironmentShift )
```

B. PARAMETER FILE

In the individual vehicle parameter file, any parameters that needed to be different than those contained in the 'standard_params.rdr' or 'macros.rdr' were over-ridden at this point. The comments at the beginning of the file provide a good explaination of the 'inheritance' method used to define vehicle behaviors and abilities. The definition of the LHA is very similar to the definition of the AAV because it was sufficient for this study. Many changes

would be necessary to truly simulate an LHA. These files are in the directory 'common/src/ModSAF/entities/':

'US_LHA1_params.rdr':

```
;; $Revision: 1.0 $
;;
;; This file defines all the model parameters for a vehicle_US_LHA1, a
;; Tarawa class amphibious assault ship.

;; Note:  There are currently three levels of specificity in the vehicle
;;      .rdr files.  They are, from the general to the specific:
;;
;;      - standard_params.rdr
;;      - macros.rdr
;;      - the individual vehicle .rdr files
;;
;;      When adding a parameter, you must determine in which of the above
;;      three files it belongs.  Generally, this set of rules may be
;;      followed:
;;
;;      - If the parameter has the same value(s) for all vehicles of a
;;        category (the categories being ground vehicles, fixed wing
;;        aircraft, rotary wing aircraft, and missiles), then it belongs
;;        in standard_params.rdr
;;
;;      - If the parameter has the same value(s) for many vehicles, but
;;        not all vehicles of a category, then make a macro out of it,
;;        put the macro in macros.rdr, and refer to the macro in the
;;        appropriate vehicle .rdr files
;;
;;      - If the parameter is different for most vehicles, then it is
;;        best to specify it in each vehicle .rdr file
;;
;;      When looking for a parameter, start at the individual vehicle .rdr
;;      file.  If it's not there, look in macros.rdr.  If it's not there,
;;      look in standard_params.rdr.


US_LHA1_MODEL_PARAMETERS {

(SM_Entity DEFAULT_DEAD_RECKONING_PARAMETERS
        (vehicle_class vehicleClassAmphibWarfare)
        (guises vehicle_US_LHA1 vehicle_USSR_BMP) ;; currently no guises
        (send_dis_deactivate true)
        )

(SM_Components (hull SM_TrackedHull SAFCapabilityMobility)
          (primary-turret [SM_GenericTurret | 0])
          (commander-sight [SM_Visual | 0])
```

```
        (driver-sight [SM_Visual | 1])
        (gunner-sight [SM_Visual | 2])
        (main-gun [SM_BallisticGun | 0] SAFCapabilityFirepower)
        (machine-gun [SM_BallisticGun | 1] SAFCapabilityFirepower)
        )

(SM_TrackedHull (soils (SOIL_DEFAULT      (max_speeds 38.0 38.0) ;24 MPH
                                          (max_decel 0.3)
                                          (max_climb 0.0)
                       SOIL_DEFAULT_TRACKED)
                (SOIL_ROAD        (max_speeds 0.0 0.0)
                                          (max_decel 0.0)
                                          (max_climb 0.0)
                       SOIL_ROAD_TRACKED)
                (SOIL_RCI250      (max_speeds 0.0 0.0)
                                          (max_decel 0.0)
                                          (max_climb 0.0)
                       SOIL_RCI250_TRACKED)
                (SOIL_RCI050      (max_speeds 0.0 0.0)
                                          (max_decel 0.0)
                                          (max_climb 0.0)
                       SOIL_RCI050_TRACKED)
                (SOIL_SHALLOW_WATER  (max_speeds 0.0 0.0)
                                          (max_decel 0.0)
                                          (max_climb 0.0)
                      SOIL_SHALLOW_WATER_TRACKED)
                (SOIL_DEEP_WATER    (max_speeds 38.0 38.0) ; 24 MPH
                                          (max_accel 1.0)
                                          (max_decel 1.0)
                                          (max_climb 0.0)
                       SOIL_DEEP_WATER_TRACKED)
            )

          (fuel_usage (0.0   100.0)
                 (0.125  12.5))
          )


;; Primary turret
([SM_GenericTurret | 0] (physdb_name "primary-turret")
              (rates continuous 0.0 40.0))

;; Main gun - MK19 Mod 3 40 mm machine gun
([SM_BallisticGun | 0]  (physdb_name "main-gun")
                (sensor_name "gunner-sight")
                (hit_obscuring_vehicles true)
                (rates 0.0 40.0)
                (magazine_size 96)
                (loading_block 96)
                LOAD_TIME_MACHINE_GUN
```

```
                (munitions (munition_US_M430
                        (round_velocity 242.0)
                        (rate 375)
                        (mass 0.4)
                        (min_range 18.0)
                        (max_range 2212.0)
                HIT_TABLE_SIMPLE_NO_LASER_KE
                TRACKTIME_TABLE_MACHINE_GUN)
                        )
                )

;; Machine  gun - Browning .50 (12.7mm) HB, M2
([SM_BallisticGun | 1]  (physdb_name "machine-gun")
                (sensor_name "gunner-sight")
                (hit_obscuring_vehicles true)
                (rates 0.0 40.0)
                (magazine_size 200)
                (loading_block 200)
                LOAD_TIME_MACHINE_GUN
                (munitions (munition_US_M33 ;; .50 cal. ball
                        (round_velocity 934.0)
                        (rate 550)
                        (mass 0.12)
                        (min_range 0.0)
                        (max_range 6700.0) ;
                HIT_TABLE_SIMPLE_NO_LASER_HE
                TRACKTIME_TABLE_MACHINE_GUN)
                        )
                )

;; Commander's sight
([SM_Visual | 0] VISUAL_APC_COMMANDER_DVO )

;; Driver's sight
([SM_Visual | 1]VISUAL_APC_DRIVER_DVO )

;; Gunner's sight
([SM_Visual | 2]VISUAL_APC_GUNNER_DVO )

(SM_Supplies (munition_Fuel 649.8);; 171 gallons, in liters.
        (munition_US_M430 864.0);; 40mm HEDP
        (munition_US_M33 1200.0) ;; .50 cal. ball
        )

(SM_VSpotter (background on)
        (sensors commander-sight driver-sight gunner-sight)
        VSPOTTER_SPECS
        )

 (SM_VTargeter
  (background on)
```

```
(munition_control_types
 (munition_US_M430
  (type ballistic)
  (tracking_data (name          main-gun)
          (is_gun          true)))
 (munition_US_M33
  (type ballistic)
  (tracking_data (name          machine-gun)
          (is_gun          true)))
 )
 (default_weapon "main-gun" munition_US_M430)
 VTARGETER_GROUND_TRACKED
 )

(SM_VAssess (background on)
       (sensors commander-sight
              driver-sight
              gunner-sight)
       (weapons ([objectDomainMask      |
              vehicleEnvironmentMask |
              vehicleClassMask      |
              vehicleFunctionMask   ]
             [objectDomainVehicle       |
              vehicleEnvironmentGround    |
              vehicleClassSPArmoredTracked |
              vehicleFunctionMainBattleTank]
             (0.0 3500.0 "main-gun" munition_US_M430)
             )
             ([objectDomainMask      |
              vehicleEnvironmentMask ]
             [objectDomainVehicle       |
              vehicleEnvironmentGround]
             (0.0 3500.0 "main-gun" munition_US_M430)
             )
             ([objectDomainMask      |
              vehicleEnvironmentMask |
              vehicleClassMask      ]
             [objectDomainVehicle   |
              vehicleEnvironmentAir  |
              vehicleClassRotaryWing ]
             (0.0 3500.0 "main-gun" munition_US_M430)
             )
             (objectDomainMask
              objectDomainLifeForm
              (0.0 2000.0 "machine-gun" munition_US_M33)
            )
             )
       (max_hits_on_target 20)
       (gunner_visual "gunner-sight")
       VASSESS_GROUND
        VASSESS_IFV_THREATS
```

```
        VASSESS_IFV_OPTIONAL
        (no_target_load
          ("main-gun" munition_US_M430)
          ("machine-gun" munition_US_M33)
         )
        )

(SM_VSearch
 (search_type ground)
 (background on)
 (turret_scanner "primary-turret" 3.0 0.2)
 (gun_scanner  "main-gun")
 (gunner_visual  "gunner-sight")
 (visual_scanners "commander-sight" "driver-sight")
 (stopped_duty_cycle1.01.0)
 (moving_duty_cycle1.00.0)
 (restrict2for"inside""none")
)

(SM_VTerrain
 (entity_period 1000)
 (avoidance_mask [
            VTERRAIN_BUILDING |
;           VTERRAIN_WATER |
            VTERRAIN_TREELINE |
            VTERRAIN_TREE |
            VTERRAIN_CANOPY |
            VTERRAIN_ENTITY |
          VTERRAIN_STEEP_AREA |
            VTERRAIN_BREACH_LANE
            ])
    (avoid_soils    SOIL_NO_GO    SOIL_SHALLOW_WATER    SOIL_RCI050
SOIL_RCI250 SOIL_ROAD SOIL_DEFAULT )
 (background on)
 (movement_threshold 500.0)
 (map_radius 1000.0)
 (entity_radius 500.0)
 (history_list_spacing 20.0)
 (num_history_list_points 50)
 (breach_obst_nominal_size 400.0))
 }
```

C. LOAD PARAMETER FILE

The newly created parameter file was added to 'modellist.rdr' which is located in 'common/src/ModSAF/entities/':

"US_LHA1_params.rdr"

D. REFERENCE PARAMETER MACRO

The individual vehicles use other macro definitions to handle any parameter values not specified in the vehicle parameter file. These macros are associated with the vehicle in the file 'models.rdr'. This file is also located in 'common/src/ModSAF/entities/':

```
("vehicle_US_LHA1"                    US_LHA1_MODEL_PARAMETERS
GROUND_STD_PARAMS)
```

E. ADD TO GUI

The graphical user interface has the available vehicles input during start-up. By adding the vehicle to 'libsrc/libunits/units.rdr' and doing a gmake, the vehicle will appear on the list of available entities:

```
("LHA1"     vehicle_US_LHA1)
```

F. ADD ICONS

There are several display options for vehicles and units. They can be shown as PVD (Plan View Display) pictures, military individual icons, military platoon icons, or military company icons. Each level has a defaulting behavior that will bring up an icon based on the vehicle role. To define a unique PVD, it was necessary to edit the file 'pvd.rdr' in the directory 'common/libsrc/libpvd' and run 'gmake':

```
PVD_SHIP_H_LHA{ ((PVD_TEAM  PVD_BLACK    PVD_HULL
                 (block     0.0  1.0  0.25 0.50
                            0.25 -1.0  -0.25 -1.0
                            -0.25  0.50 ))
                (PVD_WHITE PVD_BLACK   PVD_HULL
                (block     0.0  0.5 0.17 0.5
                     0.17 -0.5  0.0  -0.5)))
               }


PVD_SHIP_K_LHA{ ((PVD_TEAM  PVD_BLACK    PVD_HULL
                 (block     0.0  1.0  0.25 0.50
                      0.5  0.0
                           0.25 -1.0  -0.25 -1.0
                      0.0  0.0
                           -0.25  0.50 ))
                (PVD_WHITE PVD_BLACK   PVD_HULL
                (block     0.0  0.5 0.17 0.5
                     0.25 0.0
```

```
                    0.17 -0.5  0.0  -0.5
                    0.08  0.0)))
                }
;; Specific Ship vehicles
(vehicle_US_LHA1 (PVD_SHIP_H_LHA PVD_SHIP_K_LHA PVD_SHIP_K_LHA
        PVD_SHIP_K_LHA PVD_SHIP_K_LHA))
```

To specify a correct military icon for the LHA, it was necessary to edit the file 'bgrdb.rdr' in the directory 'common/libsrc/libbgrdb/' and run 'gmake':

```
PVD_US_SHIP_ICON { (PVD_ROTATE
            (PVD_DEFAULT
            ("elt RoundPoint /Hold"
           "elt RAmphibious")))
           }
;; Specific ship vehicles
(vehicle_US_LHA1  PVD_US_SHIP_ICON)
```

G. PHYSICAL DATABASE

The physical database is where the actual physical characteristics of the vehicle are input. The file 'physdb.rdr' is contained in the directory 'common/libsrc/libphysdb/' and a 'gmake' is required for changes to take effect:

```
(vehicle_US_LHA1
(16.0 125.0 15.0 0.0 -6.0 36287390.0 ; should be 32.0 245.0 30.0 0.0 -12.0
 1830.0 ;
 OPTICAL_CONTRAST_MACRO
 IMAGEI_CONTRAST_MACRO
 THERMAL_CONTRAST_MACRO
 ((primary-turret 0 true 1.06 1.06 2.5 1.80 2.30 0.80 360.0 360.0
   ((main-gun 1 true -0.25 0.5 0.4 0.0 1.1 0.0 0.0 45.0 -8.0 0)
    (machine-gun 2 true 0.25 0.5 0.4 0.0 1.66 0.0 0.0 45.0 -8.0 0))))))
```

LIST OF REFERENCES

[AASB94] Assault Amphibian School Battalion Student Handout, *AAV History, Mission, Equipment, and Organization*, Camp Pendleton California, July 1994.

[BARR83] Barrow, Robert H.,forward in *Assault From The Sea: Essays on the History of Amphibious Warfare*, ed. M.L. Bartlett, Annapolis: Naval Institute Press, 1983.

[BAUE69] Bauer, K. Jack, *Surfboats and Horse Marines: U.S. Naval Operations in the Mexican War, 1846-48*, Annapolis: United States Naval Institute, 1969.

[BOOK93] Booker, Brooks, Garrett, Giddings, Salisbury, Worley, *1993 DMSO Survey of Semi-Automated Forces*, July 30, 1993, Defense Modeling and Simulation Office (DMSO), University of Central Florida, Orlando, Florida.

[CALD93] Calder, Smith, Courtemanche, Mar, Ceranowicz, *ModSAF Behavior Simulation and Control*, Third Conference on Computer Generated Forces and Behavioral Representation, March 17-19, 1993, University of Central Florida, Orlando, Florida.

[CERA93a] Ceranowicz, Andrew Z., *Modular Semi-Automated Forces*, Modular Semi-Automated Forces: Recent and Historical Publications, May 13, 1994, Loral Advanced Distributed Simulation, Cambridge, Massachusetts.

[CERA93b] Ceranowicz, Andrew Z., *ModSAF and Command and Control*, Modular Semi-Automated Forces: Recent and Historical Publications, May 13, 1994, Loral Advanced Distributed Simulation, Cambridge, Massachusetts.

[CERA93c] Ceranowicz, Andrew Z., *ModSAF Capabilities*, Fourth Conference on Computer Generated Forces and Behavioral Representation, May 4-6, 1994, University of Central Florida, Orlando, Florida.

[CERA94] Ceranowicz, Coffin, Smith, Gonzalez, Ladd, *Operator Control of Behavior in ModSAF*, Fourth Conference on Computer Generated Forces and Behavioral Representation, May 4-6, 1994, University of Central Florida, Orlando, Florida.

[COUR95] Courtemanche, Ceranowicz, *ModSAF Development Status*, Fifth Conference on Computer Generated Forces and Behavioral Representation, May 9-11, 1994, University of Central Florida, Orlando, Florida.

[CULP92] Culpepper, Michael E., *Tactical Decision Making in Intelligent Agents: Developing Autonomous Forces in NPSNET*, Masters Thesis, March 1992, Naval Postgraduate School, Monterey, California.

[FMFM95] Fleet Marine Force Manual 1-0, United States Marine Corps, 1995

[HEAR92] Hearne, John Henry, Jr., *NPSNET: Physically Based, Autonomous, Naval Surface Agents*, Masters Thesis, September 1993, Naval Postgraduate School, Monterey, California.

[MARD83] Marder, Arthur J., *Mongol Attempts to Invade Japan, 1274, 1281.*, in Assault From The Sea: Essays on the History of Amphibious Warfare, ed. M.L. Bartlett, Annapolis: Naval Institute Press, 1983.

[MCAN94] McAndrews, Gary M., *Autonomous Agent Interactions in a Real-Time Simulation System*, Masters Thesis, September 1994, Naval Postgraduate School, Monterey, California.

[MCMS94] Marine Corps Modeling and Simulation Management Office mission statement, 1994.

[MODS95] *ModSAF Software Architecture Design and Overview Document*, Loral Advanced Distributed Simulation, Inc., April 14, 1995, Cambridge, Massachusetts.

[MODS94] *ModSAF User Manual*, Version 1.0, Loral Advanced Distributed Simulation, Inc., March 2, 1994, Cambridge, Massachusetts.

[MOHN94] Mohn, Howard L., *Implementation of a Tactical Mission Planner for Command and Control of Computer Generated Forces in ModSAF*, Masters Thesis, September 1994, Naval Postgraduate School, Monterey, California.

[POPE91] Pope, Arthur R., *The SIMNET Network and Protocols*, LADS Document No. 9120, June, 1991, LORAL Advanced Distributed Simulation, Cambridge, Massachusetts.

[ROBA94a] Robasky, Kim, *ModSAF 1.0 Developer's Course Handouts*, LADS Document No. 94017 v. 1.01, May 9, 1994, LORAL Advanced Distributed Simulation, Cambridge, Massachusetts.

[ROBA94b] Robasky, Kim, *ModSAF 1.0 Developer's Class Work Book*, LADS Document No. 94006 v. 1.01, June 24, 1994, LORAL Advanced Distributed Simulation, Cambridge, Massachusetts.

[RODG37] Rodgers, William L., *Greek and Roman Naval Warfare: A Study of Strategy, Tactics, and Ship Design from Salamis (480 B.C.) to Actium (31 B.C.*, Annapolis: Unites States Naval Institute, 1937.

[RYAN83] Ryan, Brendan P., *Aboukir Bay, 1801*, in Assault From The Sea: Essays on the History of Amphibious Warfare, ed. M.L. Bartlett, Annapolis: Naval Institute Press, 1983.

[STAN93a] *Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications*, Version 2.0, Third Draft, 28 May 1983, STRICOM, DMSO, Orlando, Florida.

[STAN93b] Stanzione, Smith, Brock, Mar, Calder, *Terrain Reasoning in the ODIN Semi-Automated Forces System*, Third Conference on Computer Generated Forces and Behavioral Representation, March 17-19, 1993, University of Central Florida, Orlando, Florida.

[STAN89] Stanzione, T., *Terrain Reasoning in the SIMNET Semi-automated Forces System*, BBN Systems and Technologies Corp., October 1989, Cambridge, Massachusetts.

[TAMB95] Tambe, Johnson, Jones, Koss, Laird, Rosenbloom, Schwamb, *Intelligent Agents for Interactive Simulation Environments*, AI Magazine, Spring 1995: 15-39.

[TECH93] Technical Manual 09674A-10/3, *Operators Manual, Assault Amphibious Vehicle*, Commandant of the Marine Corps (AREB), Washington, D.C. 20380-0001, January 1993.

[VANB93] Van Brackle, Petty, Gouge, Hull, *Terrain Reasoning for Reconnaissance Planning in Polygonal Terrain,* Third Conference on Computer Generated Forces and Behavioral Representation, March 17-19, 1993, University of Central Florida, Orlando, Florida.

[VRAB92] Vrablik, Robert G. and Calder, Robert B., *Networked Simulation of Multiple Aircraft Using Semi-Automated Forces*, Bolt Beranek and Newman, Inc., Systems and Technologies Division, Cambridge, Massachusetts, 1992.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
Dudley Knox Library Code 013 Naval Postgraduate School Monterey, CA 93943-5002	2
Chairman Ted Lewis, Code CS/Lt Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
Professor David R. Pratt, Code CS/Pr Computer Science Department Naval Postgraduate School Monterey, CA 93943	4
Professor John S. Falby, Code CS/Fa Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
Captain Thomas J. Sobey 437 Palo Verde Ave. Monterey, CA 93940	2